Titre: Title:	Application de la méta-heuristique d'optimisation par colonie de fourmis et de la technique K-OPT à l'affectation de cellules aux commutateurs
Auteur: Author:	Joseph R. Luc Fournier
Date:	2002
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Fournier, J. R. L. (2002). Application de la méta-heuristique d'optimisation par colonie de fourmis et de la technique K-OPT à l'affectation de cellules aux commutateurs [Mémoire de maîtrise, École Polytechnique de Montréal]. PolyPublie. https://publications.polymtl.ca/7040/

INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

ProQuest Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

UNIVERSITÉ DE MONTRÉAL

APPLICATION DE LA MÉTA-HEURISTIQUE D'OPTIMISATION PAR COLONIE DE FOURMIS ET DE LA TECHNIQUE K-OPT À L'AFFECTATION DE CELLULES AUX COMMUTATEURS

JOSEPH R. LUC FOURNIER
DÉPARTEMENT DE GÉNIE ÉLECTRIQUE ET DE GÉNIE INFORMATIQUE
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION
DU DIPLÔME DE MAÎTRISE ÈS SCIENCES APPLIQUÉES
(GÉNIE ÉLECTRIQUE)
SEPTEMBRE 2002

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Ce mémoire intitulé :

APPLICATION DE LA MÉTA-HEURISTIQUE D'OPTIMISATION PAR COLONIE DE FOURMIS ET DE LA TECHNIQUE K-OPT À L'AFFECTATION DE CELLULES AUX COMMUTATEURS

présenté par : FOURNIER Joseph R. Luc

en vue de l'obtention du diplôme de : Maîtrise ès sciences appliquées

a été dûment accepté par le jury d'examen constitué de :

M. Philippe Galinier, Ph.D., président

M. Samuel Pierre, Ph.D., membre et directeur de recherche

M. Alejandro Quintero, Ph.D., membre

À tous les chercheurs qui tentent d'améliorer
la Condition Humaine au moyen de la Science...

En espérant que la Recherche puisse contribuer à l'ouverture,
à la tolérance et au respect des différences entre Peuples,
assurant ainsi la pérennité et l'évolution de notre espèce
ainsi que du Patrimoine Humanitaire

REMERCIEMENTS

Je tiens évidemment à remercier Monsieur Samuel Pierre, mon directeur de recherche, non seulement pour ses conseils concernant la rédaction de mon mémoire, mais également pour sa bienveillance en ce qui concerne l'accomplissement de mon projet. Je suis aussi reconnaissant à mes collègues du Larim ayant travaillé sur le même problème que celui objet du présent mémoire et qui m'ont communiqué certains documents m'ayant permis de mener à terme mon projet. De manière plus globale, il m'appartient également de témoigner de la solidarité et de l'amitié existant entre les membres du Larim. Je suis enfin redevable et obligé envers l'ensemble des citoyens et résidents canadiens, ayant mis à ma disposition des institutions modernes, constituées à l'aide de deniers publics, sans lesquels l'accomplissement de ce mémoire n'aurait pu avoir lieu.

RÉSUMÉ

Le nombre d'usagers s'abonnant à un service de télécommunication mobile ne cesse de croître. L'intégration de nouveaux services aux réseaux mobiles accroît, par ailleurs, la demande relative aux ressources de ces réseaux. Cette diversification des services, procurant de nouveaux types d'accès à l'information, a des bénéfices économiques, intellectuels et sociaux évidents, mais apporte également de nombreux défis technologiques et conceptuels pour les concepteurs de ces réseaux. Par ailleurs, en raison des contraintes budgétaires imposées aux entreprises chargées d'implanter et de gérer les réseaux mobiles actuels et futurs, il leur convient de réduire autant que possible leurs coûts d'infrastructure et d'opérations si elles désirent rester compétitives. Un des moyens permettant de réduire ces coûts d'investissement et de gestion consiste en une affectation efficace des cellules aux commutateurs composant les réseaux mobiles des fournisseurs en question. La présente recherche porte précisément sur ce problème d'affectation des cellules aux commutateurs dans un réseau mobile.

Le problème d'affectation (unique ou double) de cellules aux commutateurs d'un réseau mobile est un problème NP-Complet. Il nous est donc nécessaire de recourir à une heuristique si nous souhaitons résoudre notre problème en un temps acceptable. Nous avons appliqué dans la présente étude la méta-heuristique relativement récente d'Optimisation par Colonie de Fourmis (OCF) au problème d'affectation, afin d'analyser son efficacité comparativement à d'autres méthodes qui avaient antérieurement été utilisées pour la résolution du même problème. Nous avons également combiné l'exécution de cette heuristique avec la technique d'optimisation locale k-opt afin de déterminer si elle apporterait une amélioration à la première méthode. Plusieurs variantes des méthodes ont été créées dans le but de trouver la meilleure alternative possible. Les algorithmes développés ont été appliqués aux deux problèmes (affectation unique et domiciliation double) décrits par les premiers travaux consacrés à leur résolution.

En comparaison avec les travaux précédemment effectués concernant le problème d'affectation unique, nous pouvons conclure que nos méthodes, ainsi que leur implémentation sont relativement efficaces, tant en ce qui concerne la qualité de la solution fournie, qu'en ce qui concerne le temps d'exécution de l'algorithme. Les solutions fournies par notre meilleure variante sont similaires à celles générées par les autres méthodes ayant donné les meilleurs résultats et servant de comparaison pour notre analyse, pour un temps d'exécution inférieur (parfois important) par rapport à ces autres méthodes. La complexité de notre variante la plus efficace est par ailleurs de l'ordre de $O(n^*n)$ lorsque les paramètres de l'heuristique sont fixés à une valeur «nulle». Il serait ainsi possible d'augmenter la taille des instances de notre problème sans accroître le temps d'exécution de l'algorithme par un facteur supérieur à n^*n, tout en obtenant des solutions ayant une «qualité similaire».

En ce qui concerne le problème de domiciliation double, nous avons effectué une implémentation simplifiée du problème, consistant à appliquer l'algorithme développé pour l'affectation unique à deux périodes de la journée en s'assurant de ne pas comptabiliser les coûts de liaison à deux reprises lorsqu'une cellule est affectée au même commutateur pour les deux périodes. Il serait par ailleurs possible d'appliquer l'algorithme à autant de périodes que l'on juge nécessaire. Notre implémentation ne tient compte que de l'affectation statique concernant les liens physiques à établir entre chacune des cellules et l'ensemble des commutateurs du réseau. Cet algorithme pourrait être utilisé afin de déterminer la meilleure configuration du réseau, en fonction du nombre d'appels connu ou anticipé à destination de chacune des cellules pour chacune des périodes modélisées de la journée. Il serait ensuite possible de créer un algorithme dynamique tenant compte de ces liens physiques existants, qui aurait pour fonction de déterminer en temps réel les moments opportuns des changements d'affectation concernant la gestion des cellules qui disposent de deux liens physiques.

ABSTRACT

The number of users of mobile telecommunication services hasn't ceased to increase in the last decade. Meanwhile, the integration of new services to mobile networks intensifies pressure on these networks' resources. This diversification of services, while procuring new types of access to information having obvious economic, intellectual and social advantages, brings numerous technological and design challenges for developers. In addition, because of budgetary restrictions imposed upon the companies responsible for the implementation and the administration of actual and future mobile networks, these providers ought to diminish as much as possible their infrastructure and operations costs in order to maintain competitiveness. One of the means by which one can reduce these infrastructure and operations costs is by assigning cells to switches in the mobile network in an efficient manner. The subject of the present research is precisely about this assignment problem of cells to switches in a mobile network.

The assignment (single or dual) problem of cells to switches in a mobile network is a NP-Complete problem. It is therefore necessary to use a heuristic if one wishes to solve the problem in a reasonable amount of time. We have applied in the present research the relatively new Ant Colony Optimization (ACO) meta-heuristic to the problem, in order to analyse its efficiency in comparison to other methods, which had previously been used to solve the same problem. We have also combined the execution of this heuristic with the local optimization k-opt technique in order to determine if it could improve the global method. Many variants of the methods have been created with the intention to find the best alternative. The algorithms have been applied to both problems (single and dual homing) described by the first studies regarding the assignment problem.

In comparison to other researches previously made concerning the single homing problem, one can conclude that our methods and their implementations are relatively efficient, regarding both the quality of the provided solutions and the execution time of

the algorithms. The solutions provided by our best variant are similar to the ones generated by the other methods that have produced the best results and used as comparison, for a smaller execution time (sometimes important) compared to these other algorithms. The complexity of our most efficient alternative is of $O(n*n)$ when all the parameters of the heuristic have a null value. It would therefore be possible to increase the size of the instances of our problem without increasing the execution time by a factor higher than $n*n$ while obtaining solutions with similar quality.

In regard to the dual homing problem, we have developed a simplified implementation, consisting in applying the algorithm developed for the single homing problem to two different periods of the day, while making sure not to count twice link costs of cells assigned to the same switch for both periods. It would be possible to apply the algorithm to more than two periods. Our implementation is static, taking only into account the physical links to create between the cells and the switches of the network. The algorithm could be used to determine the best network configuration, according to the number of known or anticipated calls of each cell for every considered period. It would afterwards be possible to create a dynamic algorithm that would take into account the existing physical links, which would determine in real time the appropriate moments for modifying the assignments of cells having two physical links, in respect of their management.

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

LISTE DES SIGLES ET ABBRÉVIATIONS

AC :	Arc Consistency
ACO :	Ant Colony Optimization
ACS :	Ant Colony System
AG :	Algorithme Génétique
AS :	Ant Systems
BT :	Backtracking
CLP :	Constraint Logic Programming
COP :	Constraint Optimization Problems
CP :	Constraint Programming
CSP :	Constraint Satisfaction Problems
FC :	Forward Checking
GT :	Generate and Test
HOL :	Heuristique d'Optimisation Locale
LA :	Look Ahead
MAC :	Maintaining Arc Consistency
NC :	Node Consistency
OCF:	Optimisation par Colonie de Fourmis
OEP :	Optimisation par Essaim Particulaire
PC :	Programmation par Contraintes
PVC :	Problème du Voyageur de Commerce
QAP :	Quadratique Assignment Problem
RT :	Recherche Taboue
SI :	Swarm Intelligence
TSP :	Traveling Salesman Problem

CHAPITRE I
INTRODUCTION

Le nombre d'usagers s'abonnant à un service de télécommunication mobile ne cesse de croître. La grande majorité de ces usagers n'a actuellement recours qu'à des services de téléphonie cellulaire. L'évolution récente s'oriente cependant vers le développement d'applications et de services permettant l'accès, au moyen de terminaux mobiles, à de nombreux services tels que ceux fournis sur Internet par exemple (multimédias, accès à l'information et aux données). L'intégration de ces services aux réseaux mobiles augmentera, sans doute, remarquablement le nombre d'usagers et entraînera nécessairement une plus grande demande des ressources de ces réseaux. Cet accroissement de services et d'accès à l'information, fournis sous diverses formes par l'intermédiaire des réseaux mobiles, a des bénéfices économiques, intellectuels et sociaux évidents, mais apportent également de nombreux défis technologiques et conceptuels pour les concepteurs de ces réseaux. Par ailleurs, en raison des contraintes budgétaires imposées aux entreprises chargées d'implanter et de gérer les réseaux mobiles actuels et futurs, dues à l'intense concurrence existante dans ce secteur, ayant pour effet de réduire leurs marges bénéficiaires, il leur convient de réduire autant que possible leurs coûts d'infrastructure et d'opérations si elles désirent rester compétitives (et même assurer leur existence). Un des moyens permettant de réduire ces coûts d'investissement et de gestion consiste en une affectation efficace des cellules aux commutateurs composant les réseaux mobiles des fournisseurs en question.

La présente recherche porte précisément sur ce problème d'affectation des cellules aux commutateurs dans un réseau mobile. Dans ce chapitre d'introduction, nous définirons d'abord quelques concepts de base qui permettront de définir le problème d'affectation de manière plus détaillée. Nous procéderons par la suite à une énumération des objectifs de la recherche, suivie d'une esquisse du plan du rapport.

1.1 Définitions et concepts de base

Plusieurs éléments concourent à permettre aux réseaux mobiles (téléphonie ou RCP – Réseaux de Communications Personnelles) de procurer des services similaires à ceux offerts par les réseaux traditionnels. Le *MTSO* (*Mobile Telephone Switching Office*), qui comprend le *MSC* (*Mobile Switching Center*) ou *commutateur*, sert notamment à relayer les appels provenant des cellules qui lui sont assignées vers l'interface du réseau conventionnel (*PSTN – Public Switched Telephone Network*). Le commutateur a pour fonction non seulement de gérer les appels qui lui sont transmis, mais également la facturation relative à ces appels ainsi que le positionnement des usagers dont il a la charge.

La *zone de couverture* d'un réseau mobile est divisée en *cellules*. Chacune de ces cellules comprend une *Station de Base* (*BTS – Base Transceiver Station*) qui a pour fonction de gérer les communications ayant lieu sur son «territoire» de couverture. Cette station est constituée d'une antenne permettant de transmettre (à l'aide d'un «*transmitter*») et de recevoir (à l'aide d'un «*transceiver*») les fréquences radio échangées entre elle et les *unités mobiles* situées au sein de sa cellule. Ce système de cellules a été conçu afin de permettre une réutilisation des mêmes *fréquences* (existant en quantité finie) dans plusieurs cellules non contiguës. Plus la densité de la population d'une région est élevée (en supposant que le nombre d'usagers est proportionnel à la population), plus la surface couverte par chacune des cellules sera petite, permettant ainsi de desservir un plus grand nombre d'usagers avec un nombre de fréquences identiques. Ces cellules sont assemblées en *grappes* (*cluster*). Les cellules contenues dans une grappe ne pourront réutiliser les fréquences employées par une autre cellule faisant partie du même groupe afin d'éviter les interférences. La Figure 1.1 montre un réseau divisé en cellules comprenant trois grappes.

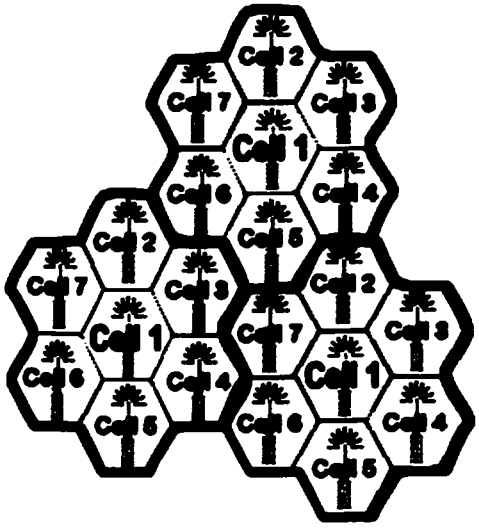

Figure 1.1 Réseau divisé en 21 cellules assemblées en 3 grappes

Les cellules comportant le même numéro peuvent utiliser les mêmes fréquences sans qu'il se produise des interférences entre les cellules concernées. Chacune de ces cellules est par ailleurs affectée à un commutateur du réseau. Ces commutateurs assurent la liaison entre le réseau mobile et les réseaux de téléphones (et autres réseaux) traditionnels. Les signaux émis par les *unités mobiles* (*Mobile Subscriber Units – MSUs*), constituées d'une *unité de contrôle* (*Control Unit*) et d'un *transmetteur* (*Transceiver*) utilisé pour établir et maintenir une communication avec les BTS des cellules, sont écoutés par l'ensemble des cellules qui se trouvent à leur proximité. Seuls les signaux atteignant une certaine intensité sont pris en compte par les cellules environnantes. L'information concernant les signaux reçus par les cellules est centralisée au niveau des commutateurs. Un commutateur, disposant ainsi de l'ensemble des informations concernant l'amplitude du signal émis par une unité mobile et reçu par l'ensemble des cellules se trouvant à sa proximité, sera en mesure de déterminer dans quelle cellule l'intensité du signal est la plus élevée et d'attribuer la gestion de l'unité mobile à la cellule appropriée.

Un problème se pose toutefois lorsque l'usager se déplace d'une cellule vers une cellule voisine. Dans la mesure où deux cellules adjacentes ne peuvent utiliser une même fréquence, il convient de transférer l'appel en cours d'une fréquence (utilisée dans la cellule dont l'usager est originaire) vers une autre fréquence (utilisée dans la cellule vers laquelle l'usager se déplace) si l'on désire ne pas mettre fin à la communication. Ce changement de fréquence se dénomme *relève* (*handoff*). La Figure 1.2 illustre le concept de relève.

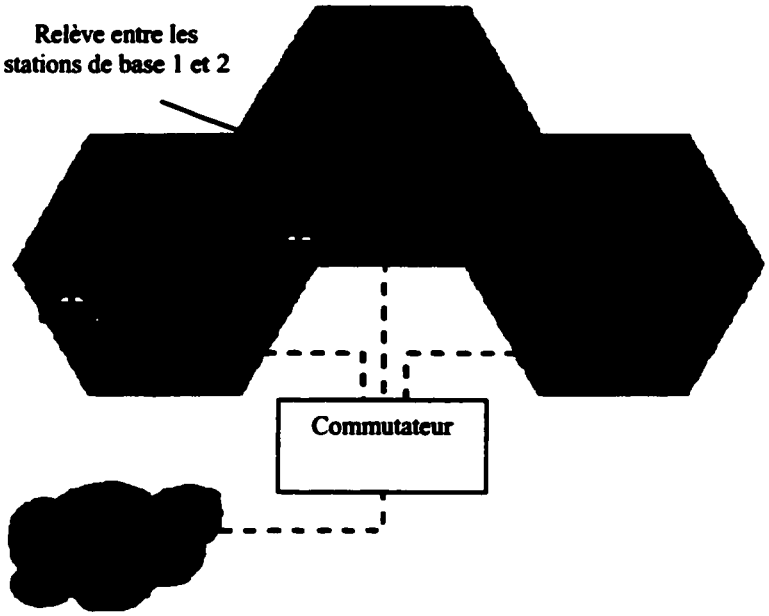

Figure 1.2 Relève entre cellules lorsque l'usager se déplace entre 2 cellules voisines

On distingue deux types de relève dans un réseau mobile : la *relève simple* qui ne fait intervenir qu'un seul commutateur, et la *relève complexe* qui exige l'intervention d'au moins deux commutateurs. La Figure 1.3 illustre la distinction entre ces deux types de relève.

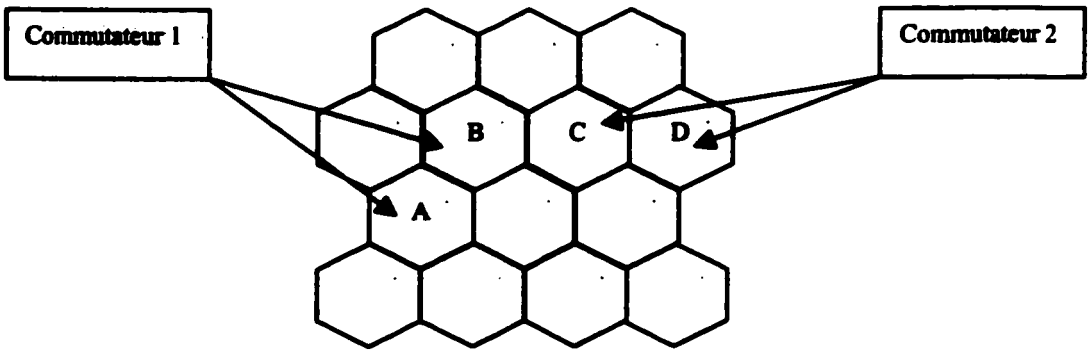

Figure 1.3 Réseau illustrant les relèves entre commutateurs

Les cellules A et B sont affectées au premier commutateur alors que le deuxième commutateur gère les signaux provenant des cellules C et D. Supposons qu'un utilisateur émettant un appel se trouve présentement au centre de la cellule B et que son appel est transféré via cette cellule au premier commutateur. Si l'utilisateur se déplace de la cellule B vers la cellule A, le premier commutateur, connaissant l'intensité relative des signaux provenant de chacune des cellules, effectuera une relève pour cet appel. Ce type de relève est relativement simple puisqu'il ne requiert que l'intervention d'un seul commutateur et n'exige aucune mise à jour dans la base de données destinées à conserver l'emplacement des usagers. Si l'usager se déplace au contraire de la cellule B vers la cellule C, le processus de relève sera plus complexe et exigera la participation des deux commutateurs. La base de données relative au positionnement des utilisateurs devra également être mise à jour suite à ce mouvement.

Un autre facteur complexifie davantage ce genre de relève. Dans l'hypothèse où les renseignements concernant la facturation de l'usager seraient compilés via le premier commutateur, il ne sera pas possible de le déconnecter de ce commutateur et l'appel devra être acheminé par l'intermédiaire des deux commutateurs en question. Il apparaît évident que la quantité de ressources réseau utilisée pour l'exécution d'une procédure relative à une relève complexe est plus importante que celle utilisée par la procédure

applicable à une relève simple. La relève complexe engendre par conséquent des coûts plus importants que la relève simple.

1.2 Éléments de la problématique

Prenant en considération un nombre de cellules et un nombre de commutateurs donnés dont l'emplacement respectif de chacun est connu, le problème consiste à affecter chaque cellule à un commutateur du réseau de manière optimale. Soulignons que chaque commutateur dispose d'une capacité finie. Le critère utilisé afin de déterminer l'optimum est le coût. L'objectif visé est de minimiser l'ensemble des coûts relatifs à l'implantation et au fonctionnement du réseau.

Notre formulation du problème considère deux composantes de coûts. Le premier est celui relatif à la liaison des cellules aux commutateurs. Le coût de relève **complexe** constitue la deuxième composante de coût prise en compte par notre modèle. Le coût de relève **simple** n'est pas directement pris en compte pour trois raisons. Ce coût est généralement négligeable comparativement au coût de relève complexe. Ce coût est par ailleurs inévitable et n'ajoute aucun élément pertinent à l'interprétation de la solution à notre problème. Chaque relève entraînera inévitablement un coût, qu'elle soit simple ou complexe. Le coût de relève simple peut être assimilé à une constante. Enfin, notre modèle en tient compte indirectement en soustrayant ce coût de relève simple au coût complexe pour chacune des relèves pouvant intervenir entre l'ensemble des cellules.

Supposons à titre d'exemple que nous disposions d'une base de données comprenant l'ensemble des coûts de relève ainsi que la fréquence des relèves entre chacune des cellules du réseau. Supposons également que les données de la base indiquent que la quantité de relèves entre les cellules C et D de la Figure 1.3 est très élevée, alors que la fréquence des relèves entre les cellules C et B est relativement faible. L'algorithme chargé de minimiser les coûts du réseau attribuera sans doute les cellules C et D au même commutateur afin de réduire les coûts de relève, alors que la cellule B pourra être affectée à un autre commutateur s'il s'avère avantageux.

Plusieurs recherches ont été effectuées sur ce problème d'affectation. Dans la mesure où ce problème est NP-Complet, ces études se sont consacrées à développer des heuristiques basées sur des approches différentes, tentant de résoudre le problème le plus efficacement possible. Merchant et Sengupta (1994, 1995) ont clairement défini le problème et ils en ont par ailleurs donné une formulation linéaire qui a été reprise dans d'autres études. Plusieurs recherches ont été effectuées au sein du *LARIM* (*Laboratoire de recherche en Réseautique et Informatique Mobile*) afin de tenter d'améliorer les solutions déjà proposées. Houéto et al. (2001a, 2001b), par exemple, ont appliqué une méthode de recherche taboue afin de résoudre le même problème. Hedible et al. (2000, 2001) ont proposé un algorithme génétique pour la résolution dudit problème. La programmation par contraintes a été utilisée à l'occasion de deux études distinctes effectuées par Amoussou et al. (2001) ainsi que par André et al (2001). Deux nouvelles modélisations du problème ont été proposées dans ces recherches. Amoussou et al. ont notamment développé une méthode exacte permettant de résoudre rapidement les problèmes de petite taille (50 cellules et 4 commutateurs). André et al. ont amélioré cet algorithme exact en diminuant son temps d'exécution et ont développé une autre méthode, incorporant la recherche taboue à la programmation par contraintes, qui a donné des résultats se rapprochant de l'optimum.

André et al. ont par ailleurs adapté leur algorithme au problème de domiciliation double sans rechercher à optimiser l'algorithme employé. La domiciliation double permet d'affecter une cellule à deux commutateurs dans l'hypothèse où une telle assignation aurait pour conséquence de diminuer la valeur de la fonction-objectif. Il peut en effet être avantageux d'attribuer une cellule à deux commutateurs, si nous nous retrouvons dans une situation où les schémas d'appels relatifs à une cellule sont notablement modifiés au cours d'une même journée par exemple.

1.3 Objectifs de la recherche

Notre étude a pour objet l'application de la technique relativement nouvelle d'Optimisation par Colonie de Fourmis (OCF : ACO – Ant Colony Optimization) au

problème d'affectation. Cette méta-heuristique est en outre combinée avec plusieurs méthodes de recherche locale dénommées 2-opt, 3-opt et Lin-Kernighan dans la littérature et décrites ultérieurement. L'heuristique développée sera appliquée au problème d'affectation unique et sera également adaptée pour la résolution de la variante du problème de domiciliation double proposée par Merchant et Sengupta (1994, 1995). L'algorithme mis en œuvre pour la résolution de cette variante pourra en outre être adapté pour la résolution de problèmes comprenant plusieurs périodes au cours d'une même journée. Il serait ainsi possible au moyen d'un algorithme dynamique, de modifier l'assignation d'une cellule à plusieurs reprises au cours d'une même journée s'il s'avérait avantageux de la faire, en tenant compte des coûts de relèves comptabilisés et des liaisons physiques existantes, qui auraient été sélectionnées à l'aide de l'algorithme statique. Nous dénombrons six principales contributions de la présente étude à la communauté scientifique :

1) Application d'une nouvelle méta-heuristique au problème d'affectation ;
2) Accroissement du champ d'application de la méthode générique d'OCF ;
3) Hybridation de la méta-heuristique avec des techniques de recherche locale ;
4) Application de l'heuristique au problème de domiciliation double ;
5) Création d'un algorithme pouvant être adapté aux problèmes comportant plusieurs périodes au cours d'une même journée ;
6) Implémentation d'un algorithme statique pouvant être utilisé conjointement avec un algorithme dynamique.

Cette étude permettra par ailleurs de mesurer les bénéfices apportés en ce qui concerne la double affectation utilisée pour la résolution du problème d'affectation des cellules aux commutateurs. Les résultats obtenus pourront donner une idée sur la pertinence de poursuivre des recherches vers d'autres variantes du problème admettant par exemple une triple affectation. Elle permettra en outre de comparer un autre

algorithme (peu nombreux) incorporant la double assignation avec ceux développés par Merchant et Sengupta (1994, 1995) d'une part et par André et al. (2001) d'autre part.

1.4 Plan du mémoire

Quatre autres chapitres suivront ce présent chapitre. Le deuxième chapitre est consacré à la description et à la modélisation de notre problème ainsi qu'à une revue de la littérature et des méthodes qui ont été utilisées pour résoudre ce problème d'affectation. Le chapitre 3 décrit la méthode préconisée dans ce mémoire pour résoudre ce problème. L'implémentation de la méthode et l'analyse de ses résultats sont effectuées au chapitre 4 en comparaison avec ceux obtenus avec d'autres techniques utilisées pour la résolution du même problème. Nous terminerons avec une conclusion qui a pour objectif d'effectuer une synthèse de la présente étude, d'énumérer les limitations de la méthode mise en œuvre et de donner quelques indications concernant les possibilités de perfectionnement éventuel de la technique utilisée.

CHAPITRE II
AFFECTATION DE CELLULES À DES COMMUTATEURS DANS UN RÉSEAU MOBILE

Il n'existe aucune méthode permettant d'obtenir avec certitude le coût d'affectation optimal en temps polynomial, pour les problèmes d'affectation de cellules aux commutateurs d'un réseau mobile de moyenne ou grande taille. Il est nécessaire de recourir à des algorithmes heuristiques afin de trouver une solution à ces problèmes en un temps acceptable. Le présent chapitre est divisé en deux sections. Une formulation du problème est présentée dans la première section. La deuxième section décrit un des algorithmes (le plus performant parmi trois) développés par Merchant et Sengupta (1994, 1995) pour la résolution de ce problème, ainsi que les trois techniques appliquées au même problème qui ont donné les meilleurs résultats à ce jour.

2.1 Formulation algébrique du problème

Merchant et Sengupta ont formulé deux variantes concernant notre problème sous forme de programme linéaire en nombres entiers. Le premier modèle n'admet qu'une seule affectation des cellules aux commutateurs alors que le deuxième admet une domiciliation double. Nous présenterons successivement ces modèles algébriques dans deux sous-sections.

2.1.1 Modèle d'affectation simple

Supposons un réseau comprenant n cellules devant être affectées à m commutateurs. Nous supposons que l'emplacement des cellules et des commutateurs est fixe et connu. Définissons la variable H_{ij} comme étant le coût par unité de temps d'une relève simple entre les cellules i et j $(i, j = 1, \ldots, n \text{ et } i \neq j)$, toutes deux attribuées au même commutateur. La variable H'_{ij} se définit comme le coût par unité de temps d'une relève complexe entre les cellules i et j, chacune étant affectée à un commutateur

différent. La valeur des variables H_{ij} et H'_{ij} varie en fonction de la quantité de relèves, que l'on suppose connue, par unité de temps entre les cellules concernées.

La variable c_{ik} se définit comme le coût d'amortissement de la liaison par unité de temps entre la cellule i et le commutateur k ($i = 1, ..., n$; $k = 1, ..., m$). Le nombre d'appels par unité de temps destiné à la cellule i est représenté par la variable λ_i. La capacité d'un commutateur est désignée par M_k.

La valeur des variables binaires x_{ik} se définissent ainsi :

$$x_{ik} = \begin{cases} 1 & \text{Si la cellule } i \text{ est affectée au commutateur } k \\ 0 & \text{Sinon} \end{cases}$$

L'affectation des cellules aux commutateurs du réseau est soumise à diverses contraintes. Chaque cellule ne doit premièrement être attribuée qu'à un seul commutateur. La formule tenant compte de cette contrainte est la suivante :

$$\sum_{k=1}^{m} x_{ik} = 1 \quad \text{Pour } i = 1, \ldots, n \tag{2.1}$$

La capacité des commutateurs ne pourra par ailleurs pas être dépassée. Cette contrainte est modélisée de la manière suivante :

$$\sum_{i=1}^{n} \lambda_i x_{ik} \leq M_k \quad \text{Pour } k = 1, \ldots, m \tag{2.2}$$

Il nous est nécessaire d'ajouter au modèle deux autres variables afin de modéliser les contraintes relatives aux relèves. Ces variables binaires z_{ijk} et y_{ij} sont définies par les contraintes suivantes :

$$z_{ijk} = x_{ik}\, x_{jk} \qquad \text{Pour } i, j = 1, \ldots, n ; \quad k = 1, \ldots, m ; \quad i \neq j \tag{2.3}$$

La valeur de z_{ijk} est égale à 1 lorsque les deux cellules i et j sont affectées au même commutateur k, alors qu'elle est de 0 dans l'hypothèse inverse.

Par ailleurs :

$$y_{ij} = \sum_{k=1}^{m} z_{ijk} \qquad \text{Pour } i, j = 1, \ldots, n ; \qquad i \neq j \tag{2.4}$$

Tout comme la variable z_{ijk}, la variable y_{ij} est égale à 1 lorsque les cellules i et j sont liées au même commutateur et à 0 lorsqu'elles sont attribuées à des commutateurs différents.

La troisième contrainte n'est cependant pas linéaire telle que formulée ci-dessus. Il nous est nécessaire de la remplacer par une série de contraintes équivalentes, comme le proposent Merchant et Sengupta (1994, 1995), si nous désirons formuler notre problème comme un modèle linéaire. Ajoutons à notre modèle les quatre contraintes suivantes :

$$z_{ijk} \leq x_{ik} \tag{2.5}$$

$$z_{ijk} \leq x_{jk} \tag{2.6}$$

$$z_{ijk} \geq x_{ik} + x_{jk} - 1 \tag{2.7}$$

$$z_{ijk} \geq 0 \tag{2.8}$$

La dernière série de contraintes concerne la non-négativité et l'intégrité des variables binaires x_{ik}.

$$x_{ik} = 0 \text{ ou } 1 \qquad \text{Pour } i = 1, \ldots, n ; k = 1, \ldots, m \tag{2.9}$$

Notons qu'il n'est pas nécessaire de définir des contraintes de non-négativité et d'intégrité en ce qui concerne les variables z_{ijk} et y_{ij} puisqu'elles sont implicitement définies par les contraintes (2.3) - remplacée par la série équivalente (2.5) à (2.8) - et (2.4).

Suite à la définition de toutes les variables et de l'ensemble des contraintes du problème, il ne nous reste plus qu'à définir la fonction-objectif qui est la suivante :

$$Min\, f = \sum_{i=1}^{n} \sum_{k=1}^{m} c_{ik}x_{ik} + \sum_{i=1}^{n} \sum_{j=1, i\neq j}^{n} H'_{ij}(1 - y_{ij}) + \sum_{i=1}^{n} \sum_{j=1, i\neq j}^{n} H_{ij}y_{ij} \qquad (2.10)$$

Le premier terme de la fonction concerne les coûts de liaison des cellules aux commutateurs. Les coûts de relève complexe et simple sont représentés par les second et troisième termes respectivement. Rappelons que les coûts de la fonction-objectif que l'on veut minimiser sont exprimés par unité de temps.

Comme il a été mentionné plus tôt, il convient de ne tenir compte du coût de relève simple que de manière indirecte. Mentionnons que ce coût est négligeable et qu'il peut être assimilé à une constante.

Définissons à cette fin la variable suivante :

$$h_{ij} = H'_{ij} - H_{ij}$$

qui correspond au coût de relève complexe réduit (par unité de temps) entre les cellules i et j.

Suite à ce dernier ajout, notre fonction-objectif devient la suivante :

$$Min\, f = \sum_{i=1}^{n} \sum_{k=1}^{m} c_{ik}x_{ik} + \sum_{i=1}^{n} \sum_{j=1, i\neq j}^{n} h_{ij}(1 - y_{ij}) \qquad (2.11)$$

2.1.2 Modèle de domiciliation double

Ce modèle considère deux périodes par jour au cours desquelles le volume d'appels des cellules et les schémas de relèves sont différents. Les définitions applicables aux variables λ_i et h_{ij} sont les mêmes, à l'exception qu'elles ne concernent qu'une période de la journée. Le modèle définit deux autres séries de variables λ'_i et h'_{ij}, ayant les mêmes significations que les précédentes (volume d'appel pour chacune des cellules et coût de relève entre cellules attribuées à des commutateurs différents), mais applicables à la deuxième période. Aucune modification n'est apportée aux variables M_k et c_{ik} correspondant respectivement à la capacité des commutateurs et au coût de liaison pour chaque cellule à chacun des commutateurs du réseau. Les séries de variables binaires x_{ik}, y_{ij} et z_{ijk}, servant à déterminer si une cellule est ou n'est pas attribuée à un commutateur, ne subissent également aucune modification mais ne s'appliquent que pour la première période. La deuxième période est représentée par les séries de variables équivalentes x'_{ik}, y'_{ij} et z'_{ijk}.

Le problème de double domiciliation est modélisé par Merchant et Sengupta comme un problème d'affectation simple, mais qui doit être effectué pour chacune des périodes comportant un schéma d'appels et un coût de relève par unité de temps différents. Le modèle ajoute cependant la nouvelle série de contraintes ci-après définie afin de s'assurer que les coûts de liaison des cellules qui sont attribuées aux mêmes commutateurs pendant les deux périodes ne soient pas comptabilisés à deux reprises.

$$w_{ik} = x_{ik} \vee x'_{ik} \qquad \text{Pour } i = 1, \ldots, n\ ;\ k = 1, \ldots, m \tag{2.12}$$

Le symbole «∨» correspond à l'opérateur logique «OU». La fonction-objectif pour ce problème de domiciliation double se définit de la manière suivante :

$$Min\, f = \sum_{i=1}^{n} \sum_{k=1}^{m} c_{ik} w_{ik} + \sum_{i=1}^{n} \sum_{j=1, i \neq j}^{n} h_{ij}(1 - y_{ij}) + \sum_{i=1}^{n} \sum_{j=1, i \neq j}^{n} h'_{ij}(1 - y'_{ij}) \tag{2.13}$$

La contrainte (2.12) n'étant pas sous forme linéaire, elle est remplacée par la série de contraintes équivalentes suivantes :

$$w_{ik} \geq x_{ik} \tag{2.14}$$

$$w_{ik} \geq x'_{ik} \tag{2.15}$$

$$w_{ik} \leq 1 \tag{2.16}$$

$$w_{ik} \leq x_{ik} + x'_{ik} \tag{2.17}$$

2.3 Heuristiques développées pour la résolution du problème d'affectation

La présente section décrit brièvement quatre méthodes qui ont été développées afin de résoudre le problème d'affectation. Elle est divisée en quatre sous-sections, étant chacune consacrée à la description d'une des techniques.

2.3.1 Heuristique développée par Merchant et Sengupta

Cette première sous-section décrit l'algorithme le plus efficace parmi les trois heuristiques développées par ces auteurs. Cette description est ici reprise non pas en raison de l'efficacité exemplaire de la technique (la méthode étant moins performante par rapport aux heuristiques décrites dans les sous-sections suivantes), mais puisqu'elle fut la première méthode appliquée à la résolution du problème d'affectation unique et au problème de domiciliation double (conjointement avec les deux autres techniques développées par les mêmes auteurs). L'affectation unique et la double affectation seront traitées au cours de deux développements consécutifs.

a) Affectation unique

Une affectation est considérée comme faisable si l'attribution (unique) des cellules aux commutateurs satisfait la capacité de l'ensemble des commutateurs du réseau. L'algorithme est divisé en deux grandes étapes.

- ***Première étape*** consacrée à la recherche d'une première solution faisable. Cette étape comporte elle-même trois sous étapes.

- ***Sous étape 1*** : Trier les cellules par ordre décroissant du volume d'appels, λ_i (soulignons que l'affectation initiale est vide).

- ***Sous étape 2*** : Attribuer de manière séquentielle chacune des cellules aux commutateurs du réseau en m étapes, pour k = 1, 2, ..., m. Étendre chaque solution partielle obtenue jusqu'à maintenant, en leur ajoutant toutes les affectations possibles de la $k$ème cellule. Les affectations possibles sont celles ne violant pas les contraintes de capacité des commutateurs. L'algorithme échoue s'il n'existe aucune affectation possible à cette étape pour la cellule en question. S'il est possible de procéder à des affectations de la cellule, retenir les b meilleures solutions obtenues (celles comportant les moindres coûts).

- ***Sous étape 3*** : Retourner la meilleure solution trouvée dans l'hypothèse où toutes les cellules auraient été attribuées à un commutateur. Retourner à la sous étape 1 dans l'hypothèse inverse.

- ***Deuxième étape*** consacrée à l'amélioration de la solution initiale obtenue à l'aide de la procédure décrite ci-dessus. Cette étape comporte elle-même quatre sous étapes, effectuées jusqu'à ce que la valeur de la fonction-objectif n'ait pu être diminuée suite à leur exécution.

- ***Sous étape 1*** : Marquer toutes les cellules comme étant non verrouillées.

- ***Sous étape 2*** : Trouver le meilleur mouvement faisable possible. Il s'agit du mouvement, parmi tous les mouvements faisables possibles, de la cellule i vers le commutateur k ayant pour effet de diminuer le plus possible la valeur de la fonction-objectif à cette étape (ou de l'augmenter le moins possible).

- ***Sous étape 3*** : Affecter cette cellule i au commutateur k en question, puis verrouiller temporairement cette cellule. Mémoriser le nouveau schéma d'affectation.

- ***Sous étape 4*** : Effectuer à nouveau les sous étapes 3 et 4 s'il est possible de trouver un mouvement respectant les contraintes de capacité. S'il n'est plus possible

d'effectuer un tel mouvement, sélectionner le schéma dont la valeur de la fonction-objectif est la moins élevée parmi tous ceux trouvés lors de ce cycle. Si cette valeur est inférieure à celle trouvée lors du cycle précédent, retourner à la première sous étape en utilisant le nouveau schéma trouvé comme solution de départ. L'algorithme s'achève à cette étape dans le cas contraire.

b) Double domiciliation

La méthode utilisée par Merchant et Sengupta afin de résoudre le problème de domiciliation double consiste à appliquer l'algorithme décrit ci-dessus pour l'affectation unique aux deux schémas du problème, correspondant chacun à une partie différente de la journée. La résolution des problèmes relatifs aux deux schémas est effectuée successivement, en prenant soin de ne comptabiliser les coûts de liaison qu'une seule fois si une cellule est affectée au même commutateur dans les deux schémas. L'algorithme comporte cinq étapes.

- ***Première étape*** : Établir deux problèmes d'affectation unique, correspondant chacun à un des deux schémas de la journée. Le premier problème est modélisé avec les variables λ_i, h_{ij}, M_k et c_{ik} alors que le deuxième est formulé en prenant les variables λ'_i, h'_{ij}, M_k et c_{ik}.

- ***Deuxième étape*** : Solutionner les deux problèmes en utilisant l'heuristique utilisée pour l'affectation unique. Q correspond au problème ayant la meilleure solution A, alors que Q' représente l'autre problème.

- ***Troisième étape*** : Désignons Q_1 comme étant un problème d'affectation unique identique au problème Q', à l'exception que si la solution A affecte la cellule i au commutateur k, le coût de liaison c_{ik} est égal à zéro dans le problème Q_1. Solutionner le problème Q_1 et représenter sa solution par A'.

- ***Quatrième étape*** : Désignons Q_2 comme étant un problème d'affectation unique identique au problème Q, à l'exception que si la solution A' affecte la cellule i au commutateur k, le coût de liaison c_{ik} est égal à zéro dans le problème Q_2. Solutionner le problème Q_2 et mémoriser sa nouvelle solution A.

- ***Cinquième étape*** : Effectuer à nouveau les étapes 3 et 4 jusqu'à ce qu'il ne soit plus possible de diminuer la valeur de la fonction-objectif. Les affectations correspondant aux solutions A et A' constituent la solution globale au problème de domiciliation double.

2.3.2 Algorithme génétique

Les algorithmes génétiques (AG) font partie du domaine de recherche plus large dénommé computation évolutive (evolutionary computing), constituant une technique du domaine de l'intelligence artificielle. Ils s'inspirent de la théorie de l'évolution biologique énoncée par Darwin. L'idée de computation évolutive a été introduite par Rechenberg dans son ouvrage appelé *Stratégies Évolutives* paru dans les années 1960. Son idée a été reprise par plusieurs chercheurs. Les AG furent imaginés par Holland (1973, 1975) et développés par lui-même avec la collaboration de certains de ses étudiants et collègues en 1975. Nous décrirons brièvement leurs principes dans un premier développement. L'application de cette technique à la résolution du problème d'affectation sera analysée dans un second développement.

a) Principes des algorithmes génétiques

En termes biologiques, les chromosomes d'un organisme vivant sont constitués de gènes, eux-mêmes formés de blocs d'ADN. Le *locus* correspond à la position du gène au sein du chromosome. Le génome est constitué de l'ensemble des chromosomes de l'organisme. La reproduction donne lieu à un croisement entre les gènes des parents afin de former de nouveaux chromosomes. Il est possible que des erreurs se produisent lors de la transmission (copie) des gènes, auquel cas nous parlons de mutation. En termes algorithmiques, la technique des AG classiques peut être divisée en cinq grandes étapes (notons que ces étapes peuvent être appliquées avec certaines variations).

- ***Première étape*** : Trouver une population initiale (aléatoirement ou de manière déterministe) constituée de n chromosomes représentant autant de solutions au problème à résoudre.

- ***Deuxième étape*** : Évaluer la valeur de la fonction-objectif (correspondant à la capacité d'adaptation des chromosomes en termes darwiniens) de chaque chromosome de la population.
- ***Troisième étape*** : Générer de nouveaux chromosomes en effectuant les quatre sous étapes suivantes jusqu'à ce que la nouvelle population soit complète.

- ***Sous étape 1*** : Sélectionner deux chromosomes parents de la population en fonction de la valeur de leur fonction-objectif (capacité d'adaptation). Notons que plus la valeur de la fonction est élevée, plus les chromosomes ont une chance d'être sélectionnés.

- ***Sous étape 2*** : Effectuer un croisement avec une probabilité prédéterminée entre les chromosomes parents choisis afin de former les nouveaux chromosomes. Si l'algorithme n'effectue aucun croisement pour ces chromosomes, copier les chromosomes parents sans modification dans la nouvelle population.

- ***Sous étape 3*** : Effectuer une mutation à l'endroit (locus) préétabli des chromosomes avec une probabilité prédéterminée.

- ***Sous étape 4*** : Intégrer les nouveaux chromosomes (ou les chromosomes parent n'ayant subi aucune modification) dans la nouvelle population.
- ***Quatrième étape*** : Remplacer l'ancienne population par la nouvelle population.
- ***Cinquième étape*** : Si la condition de terminaison de l'algorithme est satisfaite, arrêter son itération et retourner la meilleure solution de la population actuelle. Retourner à la deuxième étape dans l'hypothèse inverse.

Il existe plusieurs manières d'encoder les chromosomes d'une population, contenant chacun l'information relative à une solution à un problème donné. La méthode la plus répandue est l'encodage binaire. La solution contenue par chaque chromosome est dans ce cas représentée par une suite de variables booléennes. Chaque bit peut contenir une information différente concernant une seule caractéristique de la solution ou cette solution peut être représentée par l'ensemble des bits représentant une seule valeur. Un chromosome peut également représenter une valeur (entier, réel...) correspondant à une solution que l'on peut directement interpréter.

Le croisement a pour fonction de sélectionner certains gènes des parents afin de constituer de nouveaux chromosomes. Une manière simple d'effectuer ce croisement est de choisir au hasard un lieu (locus) de croisement des chromosomes et de copier tous les bits (en supposant que nos chromosomes sont encodés en utilisant des bits) situés à droite du lieu de sélection dans un des parents, et tous les bits situés à gauche de ce même point dans l'autre parent. Soulignons que le croisement est généralement effectué avec une probabilité qui est relativement élevée. La Figure 2.1 illustre un tel croisement.

Chromosome 1	11011 | 00100110110
Chromosome 2	11011 | 11000011110
Progéniture 1	11011 | 11000011110
Progéniture 2	11011 | 00100110110

Figure 2.1 Croisement de chromosomes

La mutation a pour objet de s'assurer que l'ensemble des solutions contenues dans une population ne tombe pas dans un optimum local. Elle est généralement appliquée avec une faible probabilité afin d'effectuer une diversification. Elle peut consister par exemple à modifier la valeur de un ou plusieurs bits des chromosomes. La sélection du ou des bits s'effectue habituellement de manière aléatoire. La Figure 2.2 illustre ce concept de mutation.

Progéniture initiale 1	1101111000011110
Progéniture initiale 2	1101100100110110
Progéniture modifiée 1	1100111000011110
Progéniture modifiée 2	1101101100110110

Figure 2.2 Mutation de chromosomes

Outre le niveau de probabilité applicable au croisement et à la mutation, la taille de la population constitue un autre paramètre important des AG qui peut avoir des conséquences sur l'efficacité de la méthode. La taille de la population détermine le nombre de chromosomes (solutions) devant être générés à chaque génération. Si sa taille est trop faible, l'algorithme ne pourra pas effectuer beaucoup de croisements et l'espace de recherche sera par conséquent réduit. Si sa taille est trop importante, l'algorithme sera ralenti.

Il existe plusieurs manières d'effectuer la sélection des chromosomes. Citons par exemple la roulette (chaque chromosome aura une probabilité d'être sélectionné qui est proportionnelle à la qualité de sa solution) ou la méthode dite élitiste dans laquelle les meilleures solutions sont transmises intactes à la génération suivante.

b) L'algorithme génétique utilisé pour la résolution du problème d'affectation

Hedible et al. (2000, 2001) ont choisi une représentation non binaire des chromosomes pour l'application de leur AG au problème d'affectation. Les schémas d'affectation s'interprètent directement des valeurs constituant les chaînes des chromosomes. Chaque entier d'un chromosome représente un des commutateurs du réseau, numérotés de 1 à m. La longueur de la chaîne des chromosomes est égale à n, le nombre de cellules. Les gènes (entiers de la chaîne), représentant les commutateurs, ne peuvent prendre comme valeur que les entiers se trouvant dans l'intervalle [1 à m]. L'absence d'une valeur dans une chaîne d'un chromosome indique que le commutateur représenté par ladite valeur n'est pas utilisé dans le schéma d'affectation proposé par ce chromosome. Cet encodage des chromosomes permet de respecter automatiquement la contrainte d'unicité d'affectation, aucun gène ne pouvant prendre plus d'une valeur à la fois. L'AG proposé par Hedible et al. est le suivant :

- ***Première étape*** : Générer la population initiale. Cette étape comprend deux sous étapes.

- ***Sous étape 1*** : Générer le premier chromosome de manière déterministe. Il s'agit d'affecter toutes les cellules du réseau au commutateur qui leur est le plus rapproché géographiquement, sans tenir compte de la contrainte de capacité.

- ***Sous étape 2*** : Générer les autres chromosomes de la population de manière aléatoire, toujours en relaxant la contrainte de capacité des commutateurs, mais en s'assurant qu'il n'existe aucun chromosome identique à un autre de la population. Cette population sans double assure une diversité et augmente la couverture de l'espace de recherche de l'algorithme.

- ***Deuxième étape*** : Appliquer les opérateurs génétiques. Cette étape est également divisée en deux sous étapes.

- ***Sous étape 1*** : Application de l'opérateur de croisement. L'algorithme sélectionne de manière aléatoire un seul lieu de croisement sur une paire de chromosomes qui sont eux aussi choisis de manière aléatoire dans la population. L'opérateur de croisement est appliqué avec une probabilité relativement élevée. Une opération d'inversion des gènes des chromosomes est effectuée lorsque la probabilité de croisement est supérieure à celle préfixée.

- ***Sous étape 2*** : Application de l'opérateur de mutation. L'algorithme sélectionne de manière aléatoire un seul lieu de mutation à l'intérieur de chaque chromosome. La probabilité associée à cette opération est très faible. Le chromosome peut rester inchangé si la nouvelle valeur générée par l'algorithme devant être intégrée au lieu de mutation est la même, que la valeur initialement contenue dans le chromosome à cet endroit.

- ***Troisième étape*** : Tout comme les deux étapes principales précédentes, deux opérations sont mises en œuvre à cette étape.

- ***Sous étape 1*** : Évaluer la fonction-objectif de chaque chromosome de la nouvelle population, représentant les coûts relatifs à la configuration du réseau représentée par chacun de ces chromosomes.

- ***Sous étape 2*** : Évaluer le respect de la contrainte de capacité des commutateurs pour chacun des chromosomes et rechercher le chromosome ayant le coût minimum dans la nouvelle population tout en respectant cette contrainte

- ***Quatrième étape*** : Former la nouvelle population en utilisant le principe de la roulette de casino, attribuant une probabilité de sélection plus élevée aux chromosomes ayant une valeur de la fonction-objectif plus faible. S'assurer d'avoir une population sans double. Notons que le meilleur élément de cette génération est automatiquement transmis à la nouvelle population.
- ***Cinquième étape*** : Retourner à la deuxième étape si le nombre de générations devant être produit n'est pas atteint. Le nombre de générations à produire est déterminé à l'avance. Dans l'hypothèse où le nombre de générations devant être générées est atteint, retourner la meilleure solution et passer au cycle suivant.
- ***Sixième étape*** : L'algorithme est effectué sur plusieurs cycles avant de s'achever. Le nombre de cycles est déterminé à l'avance. Si le nombre de cycles ayant été effectué est inférieur au nombre de cycles choisis, retourner à la première étape. L'algorithme produit en effet une nouvelle population à chaque cycle. Terminer l'exécution du programme dans l'hypothèse inverse. La meilleure solution de l'ensemble des cycles constitue la meilleure solution globale.

L'application de cette méthode à notre problème a donné de meilleurs résultats que ceux générés par l'heuristique de Merchant et Sengupta. Soulignons que Hedible et Pierre [4] ont aussi appliqué une version parallèle de leur AG.

2.3.3 Recherche taboue

La méthode actuelle de recherche taboue (RT) a été conçue par Glover (1989, 1990a, 1990b), bien que l'on puisse retracer ses origines jusqu'aux années 1970. Elle est notamment appliquée aux problèmes d'optimisation combinatoire. Son application a très tôt été mise en œuvre après son élaboration dans une étude effectuée par Skorin-Kapov en 1989 pour la résolution d'un problème d'affectation quadratique. Cette recherche a eu une influence sur l'implémentation de la méthode.

a) Principes de la recherche taboue

La RT est définie comme une méta-heuristique puisqu'elle se superpose à des techniques déjà connues. Elle tente de guider la recherche de ces méthodes traditionnelles (ou toute autre heuristique) afin qu'elles ne tombent pas dans un optimum local. Sa philosophie consiste à utiliser des méthodes de recherche locales efficaces (techniques de descentes locales) dans une perspective globale. Elle admet une détérioration temporaire des solutions dans le but d'effectuer une recherche de l'espace qui soit globale et afin de diminuer le risque d'être piégé dans un minimum local. Aussi, comme le souligne Glover (1989, 1990a, 1990b), cette méthode permet de relaxer temporairement certaines contraintes afin d'explorer des régions qui auraient été interdites en intégrant ces contraintes. Plusieurs éléments contribuent à l'efficacité de la méthode.

- ***Le mouvement :*** Afin d'effectuer un mouvement à partir d'une solution initiale ou subséquente vers un autre point, la technique de RT applique de manière itérative une heuristique destinée à la recherche d'une solution optimale locale. L'algorithme choisit la meilleure solution (valeur inférieure de la fonction-objectif) du voisinage (comprenant un ensemble de solutions candidates admissibles) présentement exploré à cette itération. Il se dirige à chaque itération vers un point ayant une meilleure solution ou détériorant la solution le moins possible. Il permet une détérioration temporaire afin de sortir des optimums locaux tout en évitant les risques de cycles en utilisant une liste taboue.

- ***La liste taboue :*** La RT conserve un historique des mouvements inverses de ceux effectués précédemment (ou de solutions déjà visitées) dans une liste qualifiée de taboue, afin d'éviter d'effectuer ces mouvements inverses (ou de se diriger vers un point déjà visité) pendant un certain nombre d'itérations. La taille de cette liste peut varier, bien que Glover ait suggéré d'utiliser une taille ayant environ sept éléments afin d'augmenter l'efficacité de la méthode. Chaque mouvement inverse (ou solution) perd son statut de tabou après un certain nombre d'itérations. Les éléments sont intégrés et quittent la liste selon une règle prédéterminée. Une règle très utilisée est la règle FIFO (First In First Out). Lorsque cette règle est appliquée, les éléments intégrés à la liste en

sortiront lorsqu'ils auront été présents dans la liste pendant un nombre de mouvements (itérations) qui est égal à sa taille. L'algorithme pourra donc avoir un cycle qui contient un nombre d'itérations qui est supérieur à la taille de la liste taboue. Il est aussi possible de rencontrer des cycles inférieurs à cette taille dans le cas où la liste contiendrait les mouvements inverses et non les solutions.

- ***Le critère d'aspiration :*** Le critère d'aspiration permet à un mouvement tabou (ou solution taboue) d'être effectué si la valeur de la fonction-objectif, suite au mouvement, satisfait une certaine condition. À titre d'exemple, un mouvement pourrait être effectué (ou il serait possible de se diriger vers un point) même si son statut est tabou s'il a pour effet de diminuer la valeur de la meilleure solution obtenue jusqu'à maintenant. La valeur pourrait aussi être déterminée arbitrairement ou il est possible de choisir un autre critère de décision.

- ***Mémoires :*** La mémoire à ***court terme*** est constitué par la liste taboue et a pour fonction d'éliminer (ou de diminuer dans le cas d'une liste contenant des mouvements) les cycles. Les mémoires à moyen et long termes permettent de comptabiliser des statistiques concernant les mouvements effectués et de «reconnaître» ainsi quels sont les meilleurs mouvements (menant à de bonnes solutions) et les zones de l'espace de recherche les moins explorées. Ces composantes de mémoires à moyen et long termes ont été proposées par Skorin-Kapov (1989).

- ***Mémoire à moyen terme :*** La mémoire à moyen terme a pour objet d'intensifier la recherche vers certaines zones. Elle compare les meilleures solutions qui ont été générées par l'algorithme afin de déterminer quels sont les attributs qui sont les plus fréquents parmi ces solutions. La mémoire considère que ces attributs, dans la mesure où ils se retrouvent fréquemment dans les meilleures solutions, sont susceptibles d'orienter la recherche vers des zones optimales. Elle intensifie sa recherche vers les solutions disposant de ces attributs jugés favorables à la création de bonnes solutions.

- ***Mémoire à long terme :*** Contrairement à la mémoire à moyen terme, la mémoire à long terme a pour objet de diversifier la recherche vers des zones qui n'ont pas donné

lieu à des recherches intensives. Afin de générer des solutions initiales, elle pénalise l'utilisation d'attributs qui se retrouvent fréquemment dans les meilleures solutions.

- ***Arrêt de l'algorithme :*** Quatre raisons peuvent intervenir afin de mettre fin à l'algorithme. Lorsqu'aucun mouvement n'est possible (tous les mouvements ou les solutions du voisinage de la solution courante sont tabous et ne répondent pas au critère d'aspiration), lorsque le nombre total d'itérations est supérieur au nombre prédéterminé ou lorsqu'aucune meilleure solution depuis la dernière trouvée n'a été rencontrée depuis un nombre *i* d'itérations (*i* étant préétabli) ou lorsque l'algorithme s'est exécuté pendant un temps égal à *t* (la valeur de *t* étant, comme les autres critères d'arrêt, choisi au préalable).

Soulignons que la méthode de RT nécessite une certaine adaptation de ses différents paramètres (détermination de l'ensemble constituant le voisinage des solutions candidates par exemple) afin d'être appliquée de manière efficace à un problème particulier.

b) La recherche taboue utilisée pour la résolution du problème d'affectation

L'adaptation de la RT au problème d'affectation de cellules a été mise en œuvre par Houèto et al. (2001a, 2001b). Ces auteurs ont divisé leur heuristique en trois composantes correspondant à la mémoire à court, moyen et long termes. L'algorithme relaxe les contraintes de capacité des commutateurs. L'unicité d'affectation des cellules est cependant respectée. Deux valeurs sont attribuées à chaque solution. La première valeur correspond au coût déterminé par la fonction-objectif, alors que la deuxième valeur prend non seulement en compte le coût de la solution mais également les pénalités associées au non respect des contraintes de capacité. La pénalité associée au non respect des capacités comprend une composante fixe et une composante qui varie en fonction du nombre d'unités non respectées par la solution. La recherche de la meilleure solution s'effectue en tenant compte de ces pénalités. L'acceptation de solutions plus coûteuses permet à l'algorithme de ne pas s'arrêter à un optimum local. L'algorithme est le suivant:

- ***Application de la mémoire à court terme*** : La composante de mémoire à court terme démarre la RT en générant une solution initiale en affectant les cellules aux commutateurs qui leur est le plus rapprochés, sans tenir compte des contraintes de capacité (si une des contraintes de capacité est violée, la valeur de la solution sera cependant supérieure à son coût). L'algorithme évalue ensuite itérativement l'ensemble des solutions du voisinage (constitué de toutes les solutions admissibles à partir de la solution présente en effectuant un mouvement) et effectue le mouvement vers la meilleure solution non taboue ou satisfaisant le critère d'aspiration (solution moins coûteuse que la meilleure trouvée jusqu'à maintenant). Une liste taboue de k éléments conserve les derniers mouvements afin de diminuer le risque de cycle. Ces mouvements redeviennent acceptables après k itérations. La partie de l'algorithme relatif à la mémoire à court terme prend fin si $kmax$ itérations sont intervenues depuis la dernière meilleure solution trouvée ou s'il n'existe plus de mouvement possible. Soulignons que l'algorithme dispose d'un mécanisme de rappel si la recherche s'éloigne trop des zones de solutions faisables.

- ***Application de la mémoire à moyen terme*** : L'algorithme effectue une intensification dans les régions considérées comme susceptibles de contenir de bonnes solutions. Une liste des dernières p (un nombre déterminé) meilleures solutions est maintenue. L'heuristique tente de diriger la recherche vers des directions non empruntées par la mémoire à court terme à partir des meilleures solutions conservées. Deux types de mouvements sont appliqués par la mémoire à moyen terme. La permutation, consistant à permuter les deux cellules du voisinage admissible diminuant le plus la valeur de la fonction-objectif, s'applique généralement aux solutions respectant les contraintes de capacité. Le deuxième type de mouvement mis en œuvre par la composante de mémoire à moyen terme consiste à déplacer une cellule d'un commutateur vers un autre, afin de rétablir la contrainte de capacité du commutateur d'origine. Ce mouvement permet de réduire la pénalité associée à la violation d'une contrainte, réduisant la valeur de la fonction-objectif de la solution.

- ***Application de la mémoire à long terme*** : La RT utilise sa composante de mémoire à long terme afin de diversifier la recherche vers des régions qui n'ont pas donné lieu à des explorations intensives. L'algorithme génère de nouvelles solutions initiales, en tenant compte des données historiques, afin de redémarrer le processus de recherche effectué par les composantes de mémoires à court et moyen terme.

Notons que Houèto et Pierre ont effectué de nombreux tests de leur heuristique en modifiant un paramètre à la fois afin d'analyser les résultats (taille de la liste taboue pour la mémoire à court terme, taille de la région d'intensification pour la mémoire à moyen terme et nombre de redémarrages pour la mémoire à long terme par exemple). Les résultats obtenus avec cette méthode sont supérieurs à ceux générés par l'AG décrit à la sous-section précédente.

2.3.4 Programmation par contrainte

La programmation par contrainte (PC – Constraint Programming) est une méthode qui généralement appliquée à la résolution de problèmes NP-Difficiles. Elle emprunte des techniques provenant des domaines de la programmation mathématique, de l'intelligence artificielle et de la recherche opérationnelle. Deux études ont déjà appliqué la PC afin de résoudre le problème d'affectation (Amoussou et al. (2001) ainsi que André et al. (2001)). Après avoir énoncé les principes de cette technique, nous décrirons dans un deuxième paragraphe l'heuristique mise en œuvre par Amoussou et al. Les simplifications apportées par André et al. à ce modèle ainsi que l'hybridation de la technique de la PC avec la RT seront successivement analysées dans un troisième et un quatrième paragraphe. Un dernier paragraphe décrira rapidement l'application des algorithmes d'André et Pierre au problème de double affectation.

a) Principes de la programmation par contrainte

Cette technique permet un développement rapide d'un programme, assure une efficacité économique en ce qui concerne sa maintenance et bénéficie souvent d'un temps d'exécution relativement court, comparativement à d'autres méthodes. La

représentation d'un problème en termes de contraintes fournit un modèle (ou programme) court et simple qui peut facilement être adapté à de nouvelles spécifications.

Sutherland (1963) fut le premier à concevoir un langage de programmation pour la spécification de contraints en 1963, intégré dans son système Sketchpad servant à l'interaction graphique. D'autres langages spécialisés dans la déclaration de contraintes furent développés très tôt par Fikes (Ref-Arf), Lauriere (Alice), Sussmann (CONSTRAINTS) et Borning (ThingLab). Ces langages offraient déjà les particularités propres à la programmation par contraintes. Tout comme pour la programmation par contraintes, la modélisation d'un problème est effectuée via la déclaration des spécifications du problème avec ces langages. La propagation des contraintes en fonction des décisions antérieures et une recherche guidée, effectuée de manière intelligente, sont également des caractéristiques représentatives de la programmation par contraintes et de ces langages. Notons que la programmation par déclaration (declarative programming) a une histoire relativement longue, pendant laquelle les langages comme LISP et Prolog ont été conçus. Cette approche est aussi à l'origine des bases de données relationnelles et du langage SQL.

La programmation logique par contrainte (dénommée CLP(X) – Constraint Logic Programming, le (X) représentant une série de contraintes primitives appartenant à la classe X) combine la dimension logique (Logic Programming), qui sert à spécifier ou déclarer un ensemble de possibilités, avec les contraintes d'un problème servant à diminuer l'espace de recherche en éliminant les alternatives non cohérentes au départ. Les contraintes sont utilisées par le programme afin de guider la recherche, en tenant compte des spécifications fournies par l'intéressé. Les contraintes peuvent par ailleurs, ne pas être représentées par des équations lorsque l'on résout un problème à l'aide de la PC. Il est néanmoins toujours possible d'utiliser les équations et inéquations formulées par la programmation mathématique traditionnelle, afin de représenter les contraintes d'un problème résolu par la PC (Jaffar et Lassez ont ainsi utilisé de telles contraintes en 1987, CLP($\Re$) représentant un problème comportant ce type de contraintes).

Afin de diminuer l'espace de recherche, la programmation par contrainte tente de réduire cet espace en propageant les contraintes d'un problème. Cette technique de propagation a été intégrée à un langage de programmation spécifique à la PC par Van Hentenryck en 1989. Cette propagation est un des éléments essentiels pour la résolution de problèmes de satisfaction de contraintes (CSP – Constraint Satisfaction Problems) et de problèmes d'optimisation par contraintes (COP – Constraint Optimization Problems). Nous décrivons dans la suite de ce développement certains éléments permettant de caractériser la PC.

- ***Domaine des variables et des contraintes*** : Une contrainte se définit comme une relation entre plusieurs variables, chacune prenant leurs valeurs dans un ensemble fini. Les contraintes réduisent le nombre de valeurs que peuvent prendre ces variables de manière simultanée.

- ***Algorithmes de recherche exhaustive*** : Il est possible d'effectuer une vérification exhaustive de toutes les assignations possibles des valeurs, contenues dans chacun des domaines de l'ensemble des variables d'un problème, afin de trouver une (ou la meilleur) solution. Toutefois, lorsque le nombre de variables est élevé, cette technique n'est évidemment pas efficace en termes de temps d'exécution. La technique de générer puis tester (GT – Generate and Test) utilise une telle recherche exhaustive. Elle génère une solution pour ensuite vérifier si elle satisfait toutes les contraintes du problème. La technique de retour arrière (BT – Backtracking) est plus efficace mais elle consomme un temps exponentiel pour résoudre un problème comportant de nombreuses variables. Elle consiste à tester la validité d'une solution toutes les fois qu'une variable est affectée, au fur et à mesure de l'avancement de l'algorithme. Lorsqu'elle se rend compte d'une incohérence, elle retourne à la dernière solution partielle cohérente et attribue une autre valeur à la variable ou passe à une autre variable. Cette méthode permet d'arrêter la recherche d'une solution partielle incohérente dès sa constatation.

- ***Techniques de cohérence*** : Il existe plusieurs techniques de cohérence ayant toutes une complexité différente. La cohérence de nœuds (NC – Node Consistency) s'applique aux contraintes unaires. La variable, représentée par un nœud, dispose d'une cohérence de

nœud si pour chaque valeur de son domaine, l'ensemble des contraintes unaires qui lui sont appliquées sont satisfaites. La satisfaction des contraintes binaires est vérifiée par la cohérence d'arcs (AC – Arc Consistency). L'arc *ij* est cohérent sur son arc si, pour chaque valeur *x* du domaine de la variable *i*, il existe une valeur *y* du domaine de la variable *j*, pour lesquelles la contrainte *Cij* est satisfaite. Soulignons qu'il existe plusieurs algorithmes permettant d'effectuer cette vérification de cohérence d'arc, notés AC-1 à AC-7, AC-3 et AC-4 étant les plus utilisés. La cohérence de nœud est dénommée cohérence unaire forte (strong 1-consistency), alors que la cohérence sur arcs est qualifiée de cohérence binaire forte (strong 2-consistency). Il existe des algorithmes permettant de vérifier une cohérence K forte (strongly K-consistent) pour K > 2. Lorsque la cohérence est de type cohérence 3 forte (strongly 3-consistent), on parle de cohérence de chemin (path consitency). Les cohérences où K > 2 sont rarement employées puisqu'ils ont généralement un temps d'exécution relativement long.

- ***Propagation des contraintes (réduction de l'espace de recherche)*** : L'algorithme de retour arrière (BT - Backtracking) discuté précédemment met en œuvre une forme de vérification de cohérence. Il teste la cohérence en ce qui concerne les variables déjà initialisées, retournant à une solution partielle antérieure lorsqu'il remarque une incohérence. Cette technique n'est cependant pas très efficace puisqu'elle ne fait aucune vérification en ce qui concerne les éventuelles incohérences futures. La méthode de vérification avancée (FC – Forward Checking) tente de prévenir les incohérences futures. Elle effectue une vérification partielle de la cohérence avec les variables futures non encore initialisées. Lorsqu'une valeur est attribuée à une variable, toutes les valeurs des variables futures produisant un conflit avec la présente «instantiation» sont temporairement retirées de leur domaine. Si le domaine d'une variable devient vide suite à cette instantiation, l'algorithme réalise immédiatement que la présente solution partielle est incohérente. Il est ainsi possible avec cette technique d'éliminer les branches de l'arbre de recherche qui mèneront à un échec plus rapidement qu'avec la méthode précédente. La dernière méthode de vérification de cohérence d'arcs consiste à effectuer une vérification, non seulement entre la variable présentement initialisée et les variables

futures, mais également entre toutes les variables futures. Cette technique, dénommée observation en avant (LA – (full) Look Ahead ou parfois dénommée MAC - Maintaining Arc Consistency), a pour effet d'éliminer encore plus d'incohérences futures (réduisant ainsi davantage l'espace de recherche), mais a aussi comme inconvénient d'augmenter la charge de travail devant être effectuée pour toutes instantiations.

- ***Ordre des variables et des valeurs choisies*** : L'ordre de traitement des variables et l'ordre d'assignation des valeurs constituent des paramètres pouvant avoir des conséquences non négligeables sur l'efficacité de la technique. L'ordre du choix des variables à initialiser peut être déterminé de manière statique ou dynamique. Lorsque l'ordre est déterminé statiquement, il n'est plus possible de le modifier lorsque l'exécution de la recherche est démarrée. Le choix de la variable à initialiser est, au contraire, déterminée au fur et à mesure de l'avancement de l'algorithme en fonction de l'état actuel de la recherche, lorsque l'ordre de traitement des variables est choisi de manière dynamique. La sélection dynamique de la prochaine variable à initialiser ne peut pas être mise en œuvre avec tous les algorithmes. L'algorithme de BT simple, ne disposant d'aucune information concernant les variables futures, ne peut mettre en œuvre une telle sélection dynamique. L'utilisation de la technique de vérification avancée, disposant d'informations concernant le domaine des variables futures, peut exercer une telle sélection. La technique la plus utilisée afin d'établir l'ordre des variables à initialiser est celle dite du premier échec (first-fail). Cette méthode suggère d'initialiser les variables avec le moins d'alternatives restantes en premier et qui sont plus susceptibles de mener à un échec. Cette technique permet à l'algorithme de se rendre compte le plus rapidement possible des solutions partielles qui mèneront inévitablement à l'échec, mettant ainsi fin à une recherche inutile. Aussi, si d'autres variables étaient initialisées avant celle offrant le moins d'alternatives, ces instantiations auraient pour conséquence de réduire davantage son domaine, pouvant ainsi mener à un échec qui aurait pu être évité. Il est donc justifié de commencer avec les variables pour lesquelles les contraintes sont les plus difficiles à satisfaire. La variable dénombrant le

plus de contraintes dans lesquelles elle est présente pourrait aussi être un critère de sélection d'ordre.

La technique la plus utilisée concernant la sélection de la valeur à attribuer à une variable suit une logique inverse à celle employée pour déterminer le choix de la prochaine variable. Le principe utilisé est celui du succès d'abord (succeed first). Lors de la sélection de la valeur à attribuer à une variable, l'algorithme tente de lui assigner la valeur qui est le moins susceptible de créer des conflits futurs. L'algorithme pourra par exemple choisir la valeur qui réduit le moins possible le domaine des variables futures.

b) Première implémentation de la PC au problème d'affectation

L'algorithme implémenté par Amoussou et al. (2001) applique l'approche «contraindre puis générer». Cette logique permet de réduire l'espace de recherche (et le nombre d'échecs) en diminuant le domaine des variables avant de procéder à la formulation de solutions. L'algorithme de séparation et évaluation (Branch and Bound) permet de trouver la meilleure solution.

Chacune des cellules du problème est représentée par une variable entière ayant un domaine correspondant au nombre m de commutateurs dans le réseau. Deux types de coût sont comptabilisés pour chacune des cellules. Le coût de liaison de la cellule au commutateur auquel elle est affectée et le coût de relève de cette cellule avec l'ensemble des cellules affectées à un autre commutateur. L'unicité d'affectation des cellules est automatiquement respectée par la modélisation du problème, la variable représentant chaque cellule ne pouvant prendre qu'une seule valeur à la fois. La contrainte de capacité pour chacun des commutateurs est exprimée par une variable ensembliste, qui a pour fonction de représenter ledit commutateur et de s'assurer du respect de sa capacité, en effectuant une sommation du volume d'appels de toutes les cellules qui lui sont affectées. Le domaine de valeur de ces variables ensemblistes correspond aux cellules qui lui sont affectées. Chaque commutateur vérifie le respect de sa capacité en diminuant sa capacité résiduelle à toutes les fois qu'une nouvelle cellule lui est affectée. Lorsque toutes les variables du problème sont fixées et que toutes les contraintes sont respectées,

le coût de la solution est calculé. Le coût de cette première solution sera considéré comme une borne supérieure devant être respectée par les autres solutions, constituant une nouvelle contrainte du problème. La valeur de cette borne sera successivement remplacée par toutes nouvelles meilleures solutions trouvées par la suite. Afin d'établir l'ordre des variables devant être initialisées, la technique du moindre regret concernant les coûts de liaison a été mise en œuvre. L'ordre est établi en fonction de la plus grande différence de coût, évaluée entre les deux plus petites valeurs de coût de liaison existant pour chaque cellule. En effectuant l'affectation le plus rapidement possible des cellules ayant une différence élevée entre leurs deux plus petites valeurs de coût de liaison, nous réduisons ainsi la possibilité de devoir comptabiliser des regrets importants dans l'avenir.

La modélisation du problème associe à chaque cellule du réseau une variable entière $Switch_i \in \{0, ..., m-1\}$ *pour* $i = 0, 1, .., n-1$. Les variables ensemblistes $Cells_j \subset \{0, ..., n-1\}$ *pour* $j = 0, 1, .., m-1$ représentant les commutateurs. Les variables $cCost[i, Switch_i]$ et $hCost[i]$ représentent respectivement les coûts de liaison et de relève. La fonction-objectif du problème est la suivante :

$$Min \sum (cCost[i, Switch_i] + hCost[i]),\ i = 0, 1, ..., n-1,\ Sous\ la\ contrainte :$$

$$\sum_{i \in Cellsj} \lambda i \leq Mj$$

Cette contrainte représentant la capacité des commutateurs devant être respectée. Les variables ensemblistes $Cells_j$ sont composées de deux sous-ensembles. Le premier est celui comprenant l'ensemble des valeurs possibles *(PossibleSet)* alors que le deuxième est composé des valeurs requises *(RequiredSet)*. Les valeurs possibles correspondent à toutes les cellules pouvant être affectées au commutateur représenté par ladite variable ensembliste. Les valeurs représentant les cellules effectivement affectées audit commutateur au cours de la recherche sont intégrées au sous-ensemble des valeurs requises. Le modèle définit par ailleurs deux classes de contraintes, *CapCoherence( )* et

BorneInfReleve( ). Les contraintes de la première classe, représentant la capacité des commutateurs devant être respectée, sont propagées lors du changement du domaine d'une des variables ensemblistes $Cells_j$. La capacité résiduelle des commutateurs dont le domaine a été modifié par l'ajout de cellules (valeurs passées de *PossibleSet* à *RequiredSet* – ou éventuellement retirées de cet ensemble), est calculée afin de s'assurer que le volume d'appels de l'ensemble des cellules qui leur sont affectées soit inférieur ou égal à leur capacité. Lorsque le volume d'appels est supérieur à la capacité d'un commutateur, l'algorithme effectue un retour-arrière. Lorsque au contraire la capacité est respectée, l'algorithme retire de l'ensemble *PossibleSet* du commutateur, les cellules ayant un volume d'appels trop élevé afin de pouvoir éventuellement lui être affecté. La deuxième série de contraintes *BorneInfReleve( )* sert à rendre la recherche plus efficace. La valeur attribuée à ces contraintes est déterminée en parcourant l'ensemble des commutateurs auxquels il est toujours possible d'affecter la cellule *i*. L'algorithme calcule ensuite pour chacun des commutateurs, la somme des coûts de relève entre la cellule *i* et les autres cellules ne pouvant plus être affectées au même commutateur. La valeur de la variable *BorneInfReleve( )* est égale à la plus petite valeur trouvée pour l'ensemble des commutateurs. L'algorithme procède à ce calcul à toutes les fois que le domaine d'une des variables $Switch_i$ est modifié. Soulignons que l'algorithme utilise également une contrainte globale *IlcNullIntersection( )* propre à la librairie ILOG Solver V.4.4 utilisée pour résoudre le problème, afin d'assurer l'unicité d'affectation des cellules.

Aussi, la prise en compte de la propagation des contraintes effectuées sur chacune des séries des variables principales du modèle $Switch_i$ et $Cells_j$ est effectuée à l'aide de divers jeux de démons. Un démon se définit comme un ensemble d'opérations devant être effectuées suite au changement du domaine d'une des variables du modèle. L'ensemble de ces procédures a pour effet de réduire (dynamiquement) l'espace de recherche de l'algorithme. La recherche effectuée sur cet espace réduit est ensuite effectuée en suivant l'ordre des variables à initialiser établi de manière dynamique en utilisant la technique du moindre regret comme expliqué précédemment. Cette technique

permet également de réduire davantage l'espace de recherche. L'algorithme affecte ensuite les cellules (choisit la valeur à attribuer aux variables) aux commutateurs disposant des ressources suffisantes qui leur sont les plus rapprochés.

Cette heuristique a donné de très bons résultats mais son temps d'exécution est malheureusement relativement long pour les problèmes de grande taille, ce qui est attribuable notamment à la complexité du modèle et non à l'efficacité de l'algorithme.

c) Simplifications apportées au modèle précédent

André et al. (2001) ont apporté plusieurs simplifications au modèle décrit au paragraphe précédent afin d'augmenter l'efficacité de l'algorithme mis en œuvre par l'heuristique. Ces auteurs citent également plusieurs éléments dont il est important de prendre en considération si l'on désire obtenir une technique efficace basée sur le PC.

Le modèle simplifié construit par André et al. définit les variables suivantes:

volDappels[i] correspondant au volume d'appels par unité de temps reçu par la cellule *i* ;

coutCab[i][k] correspondant au coût d'amortissement de liaison (câble) de la cellule *i* vers le commutateur *k* ;

coutRel[i][j] correspondant au coût de relève par unité de temps entre les cellules *i* et *j* ;

capacite[k] correspondant à la capacité du volume d'appels pouvant être géré par le commutateur *k* par unité de temps.

La contrainte d'unicité d'affectation est représentée par la variable suivante :

$Comm_i \in \{0, ..., m-1\}$ *pour* $i = 0, ..., n-1$

Le coût relatif à chacune des cellules est égal à la somme du coût de liaison de ladite cellule à son commutateur et du coût des relèves avec les autres cellules :

$$cout_i = coutCab[i][Comm_i] + \sum coutRel[i][j] \quad pour\ i = 0, ..., n-1$$

La contrainte sur la capacité des commutateurs étant la suivante :

$$\sum volDappels[i] \leq capacite[k] \qquad pour\ k = 0, ..., m-1$$

André et al. ont cherché à réduire le temps d'exécution de l'algorithme (qui était relativement important pour les problèmes de grande taille) et à améliorer la qualité de la

première solution trouvée par le programme. Ils ont intuitivement pressenti qu'il serait opportun d'établir l'ordre des variables à traiter en tenant compte non seulement du coût de câblage, comme le faisait l'algorithme initial, mais aussi les coûts de relèves.

La première simplification apportée à l'algorithme afin de réduire son temps d'exécution fut d'éliminer certaines variables redondantes du problème (variables ensemblistes $Cell_k$), les démons qui lui sont associées (*DemonCellsModifie* et *DemonSwitchsModifie*) ainsi que la contrainte *AllNullIntersection* devenue inutile. Cette modification ayant apporté une réduction importante du temps d'exécution, André et al. en ont conclu une règle applicable à la PC qui stipule que l'ajout d'une contrainte (et de la redondance) doit faire gagner plus de temps à l'exécution de l'algorithme qu'elle n'en fait perdre. Si une contrainte permet de réduire l'espace de recherche de manière à diminuer le temps d'exécution total de l'algorithme, malgré l'augmentation du travail associée à la nouvelle contrainte, il est suggéré de l'ajouter au modèle. La contrainte *BorneInfReleve* est citée par les auteurs comme exemple de contrainte ayant ce bénéfice. Cette contrainte permet d'éliminer rapidement les solutions partielles ayant une borne inférieure, calculée à partir des coûts de relèves, supérieure à la meilleure solution actuelle. Notons que l'efficacité de cette contrainte a aussi été améliorée.

Les auteurs ont par ailleurs préféré calculer les coûts totaux pour chaque cellule (coûts de relève et de liaison), représentés par la contrainte *BorneInf*, afin d'effectuer un filtrage lors de la progression de l'algorithme. La qualité de la première solution a également été accrue en utilisant ces deux types de coûts (et non plus seulement le coût de câblage comme l'algorithme d'Amoussou) afin de choisir l'ordre dans lequel les différentes valeurs sont attribuées aux variables du problème. Ces modifications ont aussi amélioré la rapidité avec laquelle cette première solution est obtenue pour n'importe quelle instance du problème. Enfin, l'ordre des variables à initialiser est toujours effectué en utilisant le critère de moindre regret, mais en sélectionnant les variables ayant un petit volume d'appels en premier afin de ne pas réduire dès le départ le domaine des variables du problème. Le critère inverse, consistant à choisir les variables ayant un volume d'appels élevé en premier, a été mis en œuvre par ces auteurs

et a donné des résultats catastrophiques. La sélection des variables ayant un volume d'appels élevé en premier a pour conséquence de réduire de manière importante le domaine des variables, forçant l'algorithme à devoir procéder à des attributions ayant un regret élevé.

d) Hybridation de la PC avec la RT

André et al. ont développé leur propre variante de la technique de recherche locale RT. Contrairement à l'algorithme de RT vue précédemment (Houéto et Pierre), leur implémentation n'autorise les changements d'affectation de cellules que lorsqu'ils respectent les contraintes du problème. Elle tente de réduire le coût global de la fonction-objectif en effectuant des changements d'affectation de cellules ayant idéalement pour effet d'améliorer la solution actuelle. Tout comme pour la RT classique, elle autorise également les «ré-affectations» qui ont pour effet d'augmenter la valeur de coût de la fonction-objectif en espérant pouvoir la réduire davantage ultérieurement. Ces mouvements, qualifiés d'échecs par l'algorithme, sont mémorisés dans une liste taboue et comptabilisés à l'aide de la variable *nb_echec*. Cette variable est «ré-initialisée» à zéro lorsque l'algorithme effectue un mouvement ayant pour effet de réduire le coût de la fonction-objectif et est incrémentée de un lorsqu'il procède au contraire à un mouvement ayant pour conséquence d'augmenter la valeur de ladite fonction. L'algorithme arrêtera sa recherche lorsque la valeur de cette variable *nb_echec* aura atteint un seuil prédéterminé par l'utilisateur. Seuls les mouvements ne faisant pas partie de la liste tabou ou diminuant la valeur de la fonction sont exécutés par l'algorithme. Le changement d'affectation apportant la plus grande diminution de coût à la fonction-objectif ou l'augmentant le moins possible est sélectionné à chaque étape de l'algorithme.

Les auteurs ont ajouté à cette première implémentation simple de la méthode de RT des composantes supplémentaires afin d'accroître l'efficacité de l'algorithme. Ils ont premièrement mis en œuvre un mouvement dénommé redistribution. Ce mouvement consiste à ré-affecter une cellule considérée comme gênante (sa présence ayant pour

effet de bloquer d'autres affectations qui auraient pour conséquence de réduire le coût de la fonction, en raison de la diminution de capacité imposée au commutateur auquel elle est présentement affectée), d'un commutateur vers un autre, afin de pouvoir affecter d'autres cellules au commutateur ainsi «libéré» de la cellule dite gênante (notons que la cellule gênante ré-affectée a habituellement un volume d'appel élevé). L'algorithme augmentera ainsi légèrement le coût de la fonction en attribuant la cellule gênante à un autre commutateur, mais le diminuera davantage à la même occasion en procédant à la ré-affectation d'autres cellules vers le commutateur «libéré». Outre ce mouvement de redistribution, les auteurs ont également incorporé à l'algorithme un mouvement dit double. Ce mouvement double permet d'effectuer par exemple deux mouvements simples qui eussent violés (du moins l'un d'entre eux) les contraintes de capacité s'ils eussent été appliqués de manière indépendante, mais qui respectent lesdites contraintes lorsqu'ils sont appliqués simultanément. Enfin, les auteurs ont aussi appliqué à leur algorithme une composante dénommée contournement, qui consiste à accepter temporairement la violation d'une contrainte de capacité (de manière similaire à la technique appliquée par Houéto et Pierre), lorsque cette violation permet de réduire la valeur de la fonction en espérant rétablir le respect de la contrainte par la suite en procédant à d'autres mouvements. Le (ou les) mouvement successif menant à la solution réalisable suite à la mise en œuvre de ce contournement devra évidemment attribuer un coût à la fonction qui soit inférieur à sa valeur antérieure afin que l'opération puisse être justifiée. Il est parfois nécessaire à l'algorithme de procéder à plusieurs mouvements avant de pouvoir rétablir le respect de l'ensemble des contraintes du problème. Il est possible par exemple que le rétablissement du respect de la contrainte de capacité d'un commutateur ne puisse être obtenu qu'en violant la contrainte d'un autre commutateur. L'algorithme déplace ainsi la violation d'une contrainte relative à un commutateur vers la contrainte d'un autre commutateur. André et al. signalent la similitude de ce phénomène avec celui des chaînes d'éjection. Cette dernière variante a donné des résultats légèrement moins économiques que l'application de mouvements doubles, bien qu'elle soit plus rapide.

L'hybridation de la PC avec la méthode de RT est effectuée en utilisant comme solutions initiales pour procéder aux recherches locales (RT), celles fournies par l'algorithme de PC. Les résultats obtenus par l'heuristique sont les meilleurs parmi toutes les techniques mises en œuvre pour la résolution de ce problème. Les auteurs ont par ailleurs exécuté leur algorithme de RT avec des solutions initiales ne provenant pas de l'algorithme de PC et ayant un coût originel plus élevé. Ils ont constaté que les résultats obtenus étaient similaires à ceux obtenus lorsque les solutions initiales de départ étaient fournies par l'algorithme de PC. Ils en ont donc logiquement conclu que l'efficacité de leur méthode était due à l'algorithme de RT et non pas à la PC.

e) PC et RT appliquées au problème de domiciliation double

Rappelons que le problème de double domiciliation est similaire au problème d'affectation unique à l'exception qu'il est possible d'affecter une cellule à deux commutateurs s'il s'avère avantageux de la faire. Il peut être justifié d'effectuer une telle double affectation si le schéma d'appels varie sensiblement au cours d'une période pour certaines cellules du réseau (Merchant et Singupta divisant la journée en deux parties). La modélisation établie par André et Pierre pour l'application de la PC au problème de double affectation est semblable à celle utilisée pour l'affectation unique à l'exception qu'il faille dédoubler l'ensemble des variables du problème afin de pouvoir affecter chacune des deux séries de variables ainsi créées à une partie de la journée. Il est aussi nécessaire de ne pas comptabiliser les coûts de liaison d'une cellule à deux reprises lorsqu'elle est affectée au même commutateur pour les deux périodes. Le modèle est le suivant :

Les variables *coutCab[i][k]* et *capacite[k]* restent inchangées ;

volDappels[i][t] correspondant au volume d'appels par unité de temps reçu par la cellule *i* au cours de la période *t* ;

coutRel[i][j][t] correspondant au coût de relève par unité de temps entre les cellules *i* et *j* au cours de la période *t* ;

$Comm_i^t \in \{0, ..., m-1\}$ *pour* $i = 0, ..., n-1$ et $t = 1, 2$, correspondant à l'affectation des cellules pour la période t ;

La contrainte sur la capacité des commutateurs étant :

$$\sum volDappels[i][t] \leq capacite[k] \quad pour\ k = 0, ..., m-1 \text{ et } t = 1, 2$$

Et le coût relatif à chacune des cellules est maintenant déterminé de la manière suivante :

$$cout_i = \begin{cases} coutCab[i][Comm_i^1] + \sum coutRel[i][j][t] \text{ si } Comm_i^1 = Comm_i^2 \\ \sum coutCab[i][Comm_i^t] + \sum coutRel[i][j][t] \end{cases}$$

Pour $i = 0, ..., n-1$ et $t = 1, 2$

Les auteurs ont (légèrement) adapté leurs algorithmes de PC et de RT précédemment décrits à ce nouveau problème de domiciliation double, afin de tenir compte des deux périodes et s'assurant de ne pas comptabiliser à deux reprises une cellule affectée au même commutateur pour les deux périodes. Les résultats obtenus pour l'ensemble des tests effectués pour cette variante du problème avec les algorithmes de PC et de RT modifiés sont moins économiques que ceux obtenus pour la variante d'affectation unique. Dans la mesure où les auteurs ont effectué une adaptation relativement simpliste de leurs méthodes, il était prévisible que les résultats obtenus seraient moins performants, les algorithmes n'ayant pas fait l'objet de raffinements et étant quelque peu mal adaptés à la résolution de cette variante.

CHAPITRE III
OPTIMISATION PAR COLONIE DE FOURMIS ET MÉTHODES DE RECHERCHE LOCALE (k-opt)

«Where, in what assembly, what council, what intellectual and moral sphere, does this spirit reside to whom all must submit ... an intelligence whose eyes are persistently fixed on the future? ... We soon shall see with what startling rapidity they are able to understand each other, and adopt concerted measures.»

Maurice Maeterlinck, La vie des abeilles[1]

Il est possible de distinguer deux grandes catégories d'approches heuristiques existantes pour la résolution de problèmes d'optimisation combinatoire tel que le nôtre. La première catégorie se caractérise par la construction d'une solution à un problème, au fur et à mesure de l'avancement de l'algorithme. Les méthodes faisant partie de cette catégorie démarrent habituellement leur recherche avec un ensemble vide, et procèdent à la construction d'une solution faisable en ajoutant successivement des éléments à la solution partielle actuellement construite en se référant à certains règles. Les heuristiques composant la deuxième catégorie d'approches sont initialisées, contrairement à leurs consœurs de la première catégorie, en fournissant à leur algorithme une solution au problème (qui n'est pas forcément économique ou qui ne respecte pas nécessairement les contraintes du problème) dès le départ. Une fois initialisés, les algorithmes en question tentent d'améliorer la solution initiale (ou éventuellement de la rendre admissible) en procédant à certains mouvements prédéfinis afin de se rapprocher le plus possible (ou d'y parvenir éventuellement) d'une solution correspondant à un optimum qui est généralement local, mais qui peut parfois être global. Les algorithmes de cette catégorie sont parfois dénommés *heuristiques d'optimisation locale*. Les méthodes d'optimisation locale 2-opt, 3-opt et Lin-Kernighan, qui consistent à troquer

[1]Maurice Maeterlinck, écrivain Belge, a écrit plusieurs livres concernant les insectes vivant en société (La vie des abeilles (1901), La vie des termites (1926) et La vie des fourmis (1930)). Le prix Nobel de Littérature lui fut décerné en 1911. Traduction d'Alfred Sutro, retrouvée à l'adresse: http://209.11.144.65/eldritchpress/mm/b.html

respectivement deux, trois ou un nombre de composants variables entre deux agents du problème, constituent des heuristiques d'optimisation locale.

Il a été démontré par expérience (Reinelt, 1994) que les heuristiques d'amélioration sont, en général, plus efficaces que les algorithmes de construction en ce qui concerne la résolution du problème de TSP, auquel notre problème s'apparente. Il est cependant nécessaire de disposer d'une solution initiale afin de pouvoir procéder à son amélioration. Les deux approches peuvent être combinées afin de créer une technique globale et générale efficace pour la résolution d'un problème de ce type, consistant à appliquer une méthode de construction pour l'obtention d'une (ou plusieurs) solution initiale, pouvant ensuite être améliorée à l'aide d'une technique d'optimisation locale. Il a aussi récemment été démontré (Johnson et McGeoch, en cours de publication) que les méthodes consistant à appliquer une légère modification (mutation) à la dernière solution obtenue (ou à la meilleure solution actuelle) avant de procéder à un nouveau cycle d'un algorithme d'amélioration, donnent de meilleurs résultats que celles fournissant itérativement à un tel algorithme d'amélioration des solutions obtenues de manière aléatoire. Une recherche confirmant cette analyse est celle effectuée par Freisleben et Merz (1996a et 1996b) dans laquelle un AG est utilisé afin de créer des solutions qui sont localement optimisées à l'aide d'une heuristique d'amélioration. Une autre étude ayant également mis en œuvre avec succès cette stratégie de combinaison d'algorithmes de construction et d'amélioration est retrouvée en (Dorigo et Gambardella, 1997).

La méta-heuristique d'OCF est une méthode qui combine les deux techniques de construction et d'amélioration. Tout comme les AG en général, cette heuristique construit à chaque itération un ensemble de solutions qui correspondent globalement à une mutation de la meilleure (ou des meilleures) solution obtenue lors de l'itération précédente ou depuis le début de l'exécution. La méthode implémentée dans la présente étude effectue en outre une hybridation de la technique d'OCF et de méthodes de recherche locale.

Le présent chapitre est divisé en trois sections. La première section décrit, en termes théoriques et généraux, la méta-heuristique d'OCF. Certaines techniques de recherche locale sont analysées dans la deuxième section. La troisième section présente l'adaptation possible de la méthode d'OCF à la résolution de problèmes dynamiques.

3.1 Éléments d'optimisation par colonie de fourmis

L'Optimisation par Colonie de Fourmis (OCF : ACO - Ant Colony Optimization) constitue, tout comme la recherche taboue, une méta-heuristique ayant habituellement pour objet la résolution de problèmes d'optimisation combinatoire considérés comme NP-Difficiles. Elle fut développée à partir d'une méthode plus simple dénommée AS (Ant Systems) créée par Marco Dorigo lors de la rédaction de sa thèse de doctorat au Dipartimento di Elettronica, Politecnico di Milano, Italy en 1992. Ce système basé sur les fourmis (Ant Algorithm) a initialement été appliqué au problème classique de recherche opérationnelle du voyageur de commerce (TSP – Traveling Salesman Problem (1992) - voir également les textes (Dorigo, Maniezzo et Colorni, 1991a, 1991b) publiés très tôt concernant ce système ainsi que (Dorigo, Maniezzo et Colorni, 1996)). L'algorithme initial présentait une efficacité comparable à celle d'autres heuristiques génériques (AG et autres algorithmes évolutifs par exemple). La méthode ne pouvait cependant se comparer favorablement, en ce qui concerne son application au problème spécifique de TSP pour les instances de grande taille, à d'autres techniques déjà utilisées et ayant fait l'objet de divers raffinements pour la résolution de ce problème en particulier. Elle a eu pour effet néanmoins de stimuler la recherche (notamment en Europe) sur des alternatives algorithmiques basées sur la même technique, débouchant sur des algorithmes génériques très performants (souvent plus performants que certaines méthodes existantes) pouvant être appliqués à de nombreux problèmes. Ces algorithmes améliorés ont depuis été appliqués avec succès à divers problèmes, tels que l'assignation quadratique, le routage de véhicules, l'ordonnancement, le routage de paquets dans des réseaux de type Internet et divers autres problèmes (Den Besten, Stützle et Dorigo, 2000 ; Di Caro G. et Dorigo, 1998 ; Gambardella et Dorigo, 2000 ; Gambardella,

Taillard et Agazzi, 1999 ; Merkle, Middendorf et Schmeck, 2000 ; Stützle et Dorigo, 1999). Fort de ce succès, la méta-heuristique d'OCF a été proposée par Dorigo et Di Caro (1999) (voir également Dorigo, Di Caro et Gambardella dans (1999)) comme technique d'encadrement générale pour l'ensemble des applications et variantes existantes de l'algorithme original. La méthode d'OCF peut donc être considérée comme le résultat d'une synthèse de divers algorithmes ayant tous la même base (AS) et fournissant les caractéristiques générales propres à une nouvelle classe d'algorithmes, afin de servir de cadre de référence pour le développement futur de nouvelles implémentations.

Nous décrirons les différentes propriétés de la méthode à l'aide de quatre sous sections. La première sous-section présentera son analogie avec le comportement des fourmis biologiques. Nous reviendrons plus en détails sur les développements historiques de la méthode et de ses extensions dans une deuxième sous-section. Nous donnerons dans une troisième sous-section une représentation et une formulation génériques à un problème hypothétique, permettant sa résolution en ayant recours à la technique d'OCF. Nous terminerons dans la quatrième sous-section en expliquant le fonctionnement de la méta-heuristique.

a) Comportement des fourmis biologiques et les systèmes d'essaim intelligent (Swarm Intelligence System)

Les techniques d'AS et l'algorithme d'OCF peuvent être catégorisées parmi les nouvelles méthodes de Système d'Essaim Intelligent (SI – Swarm Intelligence) dont la méthode semblable d'Optimisation par Essaim Particulaire (OEP) fait également partie. «L'intelligence propre aux essaims est caractérisée par un système par lequel le comportement collectif des agents rudimentaires (fourmis, abeilles, guêpes, termites...), interagissant au niveau local avec leur environnement, permet l'émergence d'un schéma cohérent fonctionnant au niveau global».[1] Il est possible de trouver une introduction à la

[1] Traduit de Swarm Intelligence : http://www.sce.carleton.ca/netmanage/tony/swarm.html et http://dsp.jpl.nasa.gov/members/payman/swarm/

technique de SI en se référant aux articles (Bonabeau E. et Théraulaz, 2000 ; Hoffmeyer, 1994 ; Ward, 1998).

La méthode d'OCF s'inspire d'une étude effectuée par Goss et al. [40] dans laquelle de vraies fourmis originaires d'Argentine (*Iridomyrmex humilis*) furent utilisées. L'expérimentation a été effectuée dans un laboratoire. Une source de nourriture fut fournie à une colonie de fourmis située dans une arène fermée. Il n'était possible cependant d'accéder à cette source de nourriture à partie du nid de la colonie, qu'en empruntant une des deux branches de différentes longueurs constituant deux ponts tel qu'illustré par la Figure 3.1.

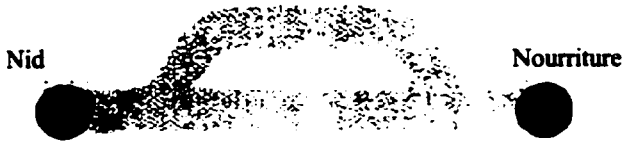

Figure 3.1 Nid et source joints par deux branches de longueurs différentes

L'observation démontre qu'après une phase transitoire, la totalité des fourmis empruntent la branche la plus courte afin de voyager entre le nid et la source et vice versa. Il est aussi constaté que la probabilité pour que la colonie choisisse la branche la plus courte augmente proportionnellement avec la différence de longueur calculée entre les deux branches. La sélection de la branche la plus courte s'explique par un phénomène auto-catalyseur (*autocatalysis*[2]) ainsi que par la différence entre la longueur des chemins ; elle se concrétise par une forme de communication indirecte dénommée «*stigmergy*», agissant sur l'environnement local. Le type de fourmis utilisé pour l'expérimentation dépose en effet sur le sol un produit chimique dénommé *phéromones*, lors de leurs déplacements (Figure 3.2).

[2] Processus par lequel une décision prise au temps t augmente la probabilité d'effectuer la même décision au temps $T > t$ - dénommé *positive feedback* par Dorigo et Di Caro [Article de Référence – nos.1 et 2]

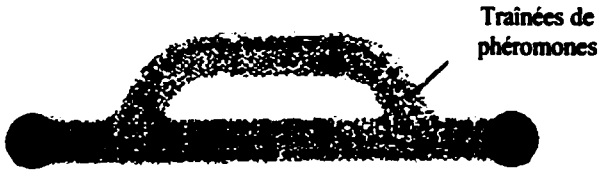

Figure 3.2 Dépôts de phéromones lors du déplacement des fourmis

Lorsqu'elles parviennent à un point d'intersection, les fourmis effectuent leur choix du chemin à emprunter en fonction d'une probabilité qui est biaisée par la quantité de phéromones ressentie se trouvant sur chacune des routes. Aucune phéromone n'étant présente sur les deux branches au début de l'expérimentation, les fourmis empruntent aléatoirement et avec une probabilité égale les deux routes possibles. Les fourmis ayant emprunté le chemin le plus court se rendent inévitablement à la source les premières. Lors de leur retour vers le nid, elles sentiront les phéromones déposées sur le chemin le plus court et l'emprunteront à nouveau avec une plus forte probabilité que le chemin le plus long. D'autres phéromones seront déposées lors de leur trajet, ce qui aura pour effet d'inciter davantage les autres fourmis de la colonie à emprunter cette route au lieu de la branche la plus longue. Ainsi, plus il y aura de phéromones déposées sur la branche la moins longue, plus les fourmis seront tentées de l'emprunter, augmentant parallèlement de plus en plus la quantité de phéromones qui y est déposée. L'ensemble de la colonie n'empruntera plus, après un certain temps, que la branche la plus courte afin d'effectuer le trajet entre le nid et la source.

Les branches de l'expérimentation relatée ci-dessus ont fait place aux arcs d'un graphe et les phéromones chimiques ont cédé leur présence aux phéromones artificielles dans les AS. Les créateurs des AS ont aussi attribué des compétences non présentes chez les fourmis naturelles afin d'accroître les capacités de leurs homologues artificielles, leur permettant ainsi de résoudre des problèmes plus complexes que ceux auxquels elles ont à faire face pour assurer leur subsistance. Ainsi, les fourmis artificielles disposent d'une mémoire leur permettant de tenir compte des contraintes d'un problème et leur donnant la faculté de faire marche arrière vers leur nid, en empruntant le même chemin utilisé pour parvenir à leur position actuelle sans commettre d'erreur. Par ailleurs, les fourmis

artificielles déposent sur leur chemin une quantité de phéromones qui est proportionnelle à la qualité de la solution rencontrée lors de la résolution d'un problème. Notons que cette propriété est également retrouvée chez certaines espèces de fourmis naturelles, qui déposent une quantité de phéromones lors de leur retour vers le nid qui est proportionnelle à la qualité de la source d'alimentation trouvée (Beckers, Deneubourg et Goss, 1993).

b) Développements historiques et extensions de la méthode

La technique initiale d'AS a été appliquée au Problème du Voyageur de Commerces (PVC). Dans l'application originale du système AS à ce problème, chaque fourmi est initialement positionnée sur une ville choisie de manière aléatoire et dispose d'une mémoire contenant la solution partielle construite jusqu'à présent (la solution n'étant constituée que de la ville de départ lorsque l'algorithme démarre). Les fourmis se déplacent itérativement de ville en ville au cours de l'exécution de l'algorithme. Lorsqu'une fourmi k se situe sur la ville i, elle choisit de se déplacer vers la ville j non encore visitée, avec une probabilité qui est déterminée par la fonction suivante :

$$p^k_{ij}(t) = \frac{[\tau_{ij}(t)]^\alpha * [\eta_{ij}]^\beta}{\Sigma_{l \in \mathbf{N}^k_i}[\tau_{il}(t)]^\alpha * [\eta_{il}]^\beta} \qquad \text{if } j \in \mathbf{N}^k_i \qquad (3.1)$$

où $\eta_{ij} = 1/d_{ij}$ est une donnée fournie par l'heuristique, α et β sont des paramètres qui ont pour objet de déterminer l'influence relative des phéromones et des valeurs de l'heuristique et $\mathbf{N}^k_i$ constituant le voisinage admissible de la fourmi k, composé de l'ensemble des villes non encore visitées par ladite fourmi. Les paramètres α et β influencent l'exécution de l'algorithme de la manière suivante : Lorsque $\alpha = 0$, la probabilité de sélection est proportionnelle à $[\eta_{ij}]^\beta$, ce qui implique que les villes les plus rapprochées de i auront plus de chance d'être choisies. Dans l'hypothèse extrême inverse où $\beta = 0$, seules les phéromones auront une influence sur les sélections effectuées par les fourmis, ce qui aura pour conséquence de diriger rapidement l'ensemble des fourmis vers des solutions stagnantes et habituellement sous optimales.

La construction des solutions se termine lorsque l'ensemble des fourmis ont achevé leur tour (constituant chacun une séquence de longueur *n, où n* = nombre de nœuds). L'algorithme met ensuite à jour les traînées de phéromones relatives aux nouveaux tours en appliquant la formule suivante :

$$\tau_{ij}(t+1) = (1-\rho) * \tau_{ij}(t) + \sum_{k=1}^{m} \Delta\tau_{ij}^{k}(t) \qquad \forall(i,j) \tag{3.2}$$

où $0 < \rho \leq 1$, correspond au taux d'évaporation (diminution) des traînées de phéromones actuelles. Avant de mettre à jour les nouvelles traînées de phéromones, l'AS applique en effet une diminution aux traînées existantes à l'aide d'un facteur constant, correspondant à un phénomène d'évaporation du produit chimique antérieurement déposé par les fourmis lors des cycles précédents. Cette évaporation évite d'accumuler des quantités trop importantes de phéromones sur les arcs et permet à l'algorithme «d'oublier» les mauvaises décisions effectuées antérieurement. Lorsque certains arcs ne sont plus utilisés par les fourmis, la quantité de phéromone qui leur est associée diminuera de manière exponentielle à chaque itération de l'algorithme. Une fois cette diminution appliquée, chaque fourmi (au nombre de *m*) déposera à nouveau des phéromones sur l'ensemble des arcs faisant partie de leur tour respectif. La quantité $\Delta\tau_{ij}^{k}(t)$ de phéromones déposée par une fourmi *k* est calculée de la manière suivante :

$$\Delta\tau_{ij}^{k}(t) = \begin{cases} 1/L^{k}(t) & \text{si l'arc } (i,j) \text{ est utilisé par la fourmi } k \\ 0 & \text{sinon} \end{cases} \tag{3.3}$$

où $L^{k}(t)$ correspond à la longueur du tour considéré de la fourmi *k*. Plus le tour est court, plus les arcs qui y sont associés recevront une quantité de phéromones importante comme il est possible de le constater en analysant l'équation (3.3). De manière plus générale, les arcs fréquemment utilisés par les fourmis et faisant partie de tours économiques recevront une quantité de phéromones relativement importante et seront ainsi plus susceptibles d'être sélectionnés lors des itérations futures.

Dans la mesure où cet algorithme (AS) initial n'était pas aussi efficace pour résoudre le problème classique de TSP comparativement à d'autres méthodes spécialement développées pour ce problème, d'autres recherches ont été effectuées afin d'améliorer sa performance. La stratégie élitiste, utilisée par les AG, a été appliquée à la méthode d'AS dans une première modification de la technique afin de tenter d'augmenter son efficacité (Dorigo, 1992 ; Dorigo, Maniezzo et Colorni, 1996). Cette modification consistait à appliquer aux arcs faisant partie de la meilleure solution obtenue jusqu'à un moment donné, une quantité de phéromones largement supérieure aux autres traînées. L'équation (3.3) était ainsi remplacée par la suivante, en ce qui concerne son application aux arcs composant la meilleure solution :

$$\Delta\tau_{ij}^{gb}(t) = \begin{cases} e/L^{gb}(t) & \text{si l'arc } (i,j) \in T^{gb} \\ 0 & \text{sinon} \end{cases} \qquad (3.4)$$

où gb = global best et e est un entier positif. Les arcs du meilleur tour T^{gb} obtenu jusqu'à présent se verront donc attribuer la quantité supplémentaire $e*1/L^{gb}$ de phéromones, L^{gb} correspondant à la longueur dudit meilleur tour.

Une autre amélioration de l'algorithme initial d'AS constitue une extension de la stratégie d'élitisme. Dénommée par ses auteurs AS_{rank} (Bullnheimer, Hartl et Strauss, 1999), elle consiste à trier (en ordre croissant) les tours construits par les fourmis à chaque itération en fonction de leur longueur, et à permettre aux seules fourmis obtenant les (w - 1) meilleurs tours de déposer des phéromones sur leur tour respectif (ainsi que sur le meilleur tour obtenu jusqu'à ce moment). L'ACS (Ant Colony System), appliqué dans (Dorigo et Gambardella, 1997a et 1997b ; Gambardella et Dorigo, 1996), est une autre variante de la technique AS qui tient compte davantage des informations collectées par les fourmis au cours des itérations précédentes en ce qui concerne l'espace de recherche. Deux mécanismes sont mis en œuvre dans cette technique. Une stratégie élitiste est premièrement appliquée afin de mettre à jour les traînées de phéromones. Seule la fourmi ayant obtenu la meilleure solution est autorisée à mettre à jour les traînées en fonction d'une règle similaire à celle utilisée pour les AS qui est la suivante :

$$\tau_{ij}(t+1) = (1-\rho) * \tau_{ij}(t) + \rho * \Delta\tau_{ij}^{best}(t) \qquad (3.5)$$

La meilleure solution peut être la meilleure obtenue lors de la présente itération (*iteration-best*) ou la meilleure obtenue depuis le début (*global-best*) de la recherche. Le deuxième mécanisme mis en œuvre par la variante consiste à choisir le prochain déplacement de chacune des fourmis en utilisant une règle dénommée «règle pseudo-aléatoire proportionnelle» (*pseudo-random proportional rule*). La règle est la suivante :

Chaque fourmi se déplace

- Soit vers la ville j pour laquelle le produit de la quantité de phéromones multipliée par les informations données par l'heuristique est maximal, avec une probabilité égale à q_0 (la formule appliquée étant la suivante : $j = arg\ max_{j \in \mathbf{N}^k_i}\{\tau_{ij}(t) * \eta_{ij}^{\beta}\}$),

- Soit en effectuant une exploration biaisée de la même manière que dans un AS «classique» avec une probabilité égale à $1 - q_0$ (la formule à appliquer dans cette hypothèse étant la même que définie à l'équation (3.1)).

La valeur q_0 est un paramètre variable. Lorsqu'il se rapproche de 1 (la plupart du temps dans les ACS), l'algorithme exploite davantage les informations déjà recueillies lors des itérations précédentes, alors que l'exploration de l'espace est semblable à la technique d'AS «classique» lorsque sa valeur se rapproche de 0.

Une dernière variante à la technique AS que nous commentons est celle dénommée *MAX-MIN Ant System* (*MMAS*) (Stützle et Hoos, 1997; Stützle, 1999; Stützle et Hoos, 2000). Cette technique est similaire à la précédente, à l'exception qu'elle ajoute des bornes minimale et maximale aux traînées de phéromones pouvant être déposées. La quantité de phéromones déposée sur les arcs ne pourra ainsi se situer à l'extérieur de l'intervalle $[\tau_{min}, \tau_{max}]$ ($\tau_{min} \leq \tau_{ij} \leq \tau_{max}, \forall \tau_{ij}$).

Notons enfin que la technique d'OCF a récemment été appliquée à des problèmes dynamiques dans lesquels les données du problème varient au cours de l'exécution du programme. Elle a été mise en œuvre pour la résolution d'un problème de routage pour un réseau de téléphone conventionnel (circuit-switched network), appliquée à une

version simulée du réseau de British Telecom. L'algorithme dénommé *AntNet* proposé par Di Caro et Dorigo (Di Caro et Dorigo, 1997 ; Di Caro et Dorigo, 1998a, 1998b et 1998c), a été appliqué au problème de routage de paquets (packet-switched network) dans un réseau de type Internet et s'est montré plus performant que de nombreuses méthodes utilisées dans ce domaine.

c) Représentation et formulation d'un problème résolu à l'aide de la technique d'OCF

Tout comme pour la programmation par contrainte, il est possible avec la technique d'OCF de formuler une représentation d'un problème qui soit très simple et intuitive. Supposons par exemple le problème de minimisation (S, f, Ω), où S est un ensemble de solutions candidates, f est la fonction-objectif qui assigne à chaque solution candidate $s \in S$ une valeur de coût $f(s, t)$ – le paramètre t indiquant que la fonction-objectif peut être fonction du temps – et Ω représentant un ensemble de contraintes. Le but recherché est de trouver une solution optimale globale $s_{opt} \in S$ (ou plus exactement, se rapprochant le plus possible de l'optimum global) satisfaisant l'ensemble des contraintes Ω du problème. Un modèle permettant d'être résolu à l'aide de l'OCF dispose d'un ensemble fini de composants $C = \{c_1, c_2, \ldots, c_n\}$ donné, les états d'un problème étant définis en fonction de séquences $x = \langle c_i, c_j, \ldots \rangle$ sur cet ensemble C. L'ensemble de toutes les séquences possibles est représenté par χ alors que l'ensemble fini des contraintes Ω détermine les états possibles $_x$, où $_x \subseteq \chi$.

En fonction de cette représentation, les fourmis procèdent à la construction de solutions en se déplaçant sur un graphe $G = (C, L)$, C correspondant aux nœuds et L à l'ensemble des «connecteurs» assurant la connexion de tous les composants C du problème. Les contraintes Ω du problème font partie des règles de construction suivies par les fourmis, autorisant ainsi une certaine flexibilité. Ces règles pourraient par exemple permettre aux fourmis de construire temporairement (si nécessaire) des solutions non faisables (comportant éventuellement une pénalité) ou les en empêcher. Les composants $c_i \in C$ ainsi que les connecteurs $l_{ij} \in L$ peuvent se voir attribuer une

traînée de phéromones τ (τ_i si elles sont associées aux composants et τ_{ij} si elles sont attribuées aux connecteurs) représentant une mémoire à long terme concernant la recherche effectuée par les fourmis. Ces composants et connecteurs peuvent également se voir attribuer des valeurs η (η_i ou η_{ij} selon le cas) correspondant à des informations heuristiques prédéterminées de l'instance d'un problème (ou déterminées par des éléments indépendants des fourmis lors de l'exécution d'un programme dynamique). Ces variables η représentent souvent des coûts pris en compte par les fourmis lors de l'extension de leurs solutions partielles, leur permettant de se déplacer de manière probabiliste sur le graphe G du problème. Chaque fourmi k dispose en outre d'une mémoire M^k lui permettant de stocker les informations concernant le chemin suivi jusqu'à présent. Cette mémoire leur permet non seulement de construire et d'évaluer une solution partielle ou finale mais également de retracer leur route empruntée afin d'y déposer des phéromones. Chaque fourmi se voit également attribuer un état de départ x_s^k ainsi qu'une ou plusieurs conditions de terminaison e^k.

d) Fonctionnement de la méta-heuristique

La méta-heuristique d'OCF est une technique partiellement stochastique qui a recours à des fourmis artificielles pour effectuer la construction de solutions de manière probabiliste. Chaque fourmi d'une colonie construit une solution finale en ajoutant itérativement des composants à sa solution partielle en tenant compte des informations relatives à l'instance du problème devant être résolue, et des traînées de phéromones artificielles qui sont modifiées de manière dynamique au cours de l'exécution du programme en fonction des expériences acquises par les fourmis lors de leur recherche.

Lorsqu'une fourmi se trouve dans l'état $x_r = \langle x_{r-1}, i \rangle$, elle tente de se déplacer vers un nœud j contenu dans son voisinage faisable $\mathbf{N}^k_i$, correspondant à l'état $\langle x_r, j \rangle \in \chi$. S'il n'est pas possible d'effectuer un tel déplacement, la fourmi se déplacera vers un nœud faisant partie de son voisinage n'admettant pas une solution faisable $_{\mathbf{IN}}{}^k_i$ générant ainsi un état non faisable $\langle x_r, j \rangle \in \chi$ mais $\langle x_r, j \rangle \notin \chi$. Les fourmis choisissent leurs déplacements en appliquant une règle probabiliste qui est fonction de la quantité de

phéromones et des informations fournies par le modèle au niveau local, leur propre histoire contenue dans leur mémoire M^k et les contraintes du modèle. La construction d'une solution se termine lorsque la condition d'achèvement e^k est satisfaite. Lorsqu'une fourmi ajoute un composant c_j à une solution partielle, elle peut instantanément mettre à jour la traînée de phéromones associée à sa solution, cette procédure étant dénommée *mise à jour en ligne pas-à-pas* (*online step-by-step pheromone update*). Elle peut aussi décider d'attendre de compléter sa solution finale avant d'effectuer un retour arrière pour déposer des phéromones, implémentant ainsi une *mise à jour retardée* (*online delayed pheromone update*).

Notons qu'une fourmi se déplace simultanément et indépendamment de ses consœurs, et qu'il lui est possible de construire une solution en ayant recours à ses seuls moyens mais qui sera généralement de piètre qualité. Les solutions de bonne qualité sont habituellement obtenues à l'aide des interactions collectives intervenant entre l'ensemble des fourmis de la colonie. Cette interaction est effectuée via une communication indirecte mise en œuvre par la création de variables globales servant à stocker l'information relative à la quantité de phéromones contenue sur tous les arcs du graphe, auxquelles toutes les fourmis de la colonie ont accès. L'acquisition de connaissances par les fourmis au cours de leur recherche, en ce qui concerne le graphe représentant le problème, influence leur comportement lors des itérations futures d'une manière similaire au phénomène de renforcement d'apprentissage (*reinforcement learning* (Sutton et Barto, 1998)). Les fourmis mettent ainsi en œuvre un processus d'apprentissage distribué par lequel elles ne s'adaptent pas d'elles-mêmes, mais modifient la manière dont le problème est perçu par la colonie dans son ensemble. Les dépôts de phéromones effectués lors de la construction par les fourmis de solutions partielles ou totales au problème d'optimisation influenceront leur propre comportement au cours des itérations futures.

L'aspect partiellement stochastique de la méthode permet aux fourmis de construire un nombre relativement important de solutions variées, leur permettant d'effectuer une exploration diversifiée et large de l'espace de recherche. Rappelons

également le mécanisme d'évaporation des traînées de phéromones mis en œuvre par l'algorithme. Cette évaporation est nécessaire afin d'éviter une convergence trop rapide de l'algorithme vers des régions sous-optimales (minimaux locaux). En appliquant ce mécanisme, les fourmis auront plus de chance d'explorer des nouvelles zones du graphe. L'utilisation des informations connues de l'instance du problème permet de diriger la recherche vers des zones qui semblent à priori prometteuses.

Terminons enfin l'explication du fonctionnement de l'algorithme en mentionnant un dernier élément qui peut être utilisé par la méthode afin d'augmenter son efficacité. Il s'agit de l'utilisation de démons (*daemon actions*). L'algorithme a recours à des démons afin de mettre en œuvre certaines actions qui ne peuvent pas être gérées par les fourmis individuellement. Citons comme exemples d'actions entreprises par «d'autorité centrale», l'activation d'une procédure d'optimisation locale ou la collecte d'informations globales pouvant être utilisées afin de déterminer s'il est opportun de déposer des phéromones supplémentaires sur certains composants, pour réorienter la recherche vers des régions qui n'eussent pas été choisies en appliquant des règles ne tenant compte que d'une perspective locale. L'algorithme pourrait par exemple décider de déposer des phéromones supplémentaires sur le chemin emprunté par la fourmi ayant construit la meilleure solution lors d'une itération. Les mises à jour des traînées de phéromones effectuées par un démon sont qualifiées de mise à jour non en-ligne (*off-line pheromone updates*). Le pseudo-code de l'algorithme général de la méta-heuristique d'ACO est illustré dans la Figure 3.3.

```
Procedure OCF Metaheuristique
    ActivitesSchedulees
        GererActivitesDesFourmis ()
        EvaporerPheromones ()
        AppliquerActionsDesDemons ()
    Fin ActivitesSchedulees
Fin OCF Metaheuristique
```

Figure 3.3 Méta-heuristique générique d'OCF

Tel qu'illustré à la Figure 3.3, la fonction globale *ActivitesSchedulees* gère les trois principales composantes de l'algorithme : l'activité des fourmis, l'évaporation des phéromones et l'application des éventuels démons.

3.2 Recherche locale

Plusieurs méthodes de recherche locale (souvent «gourmandes» - greedy) ont été imaginées et appliquées aux problèmes d'optimisation combinatoire. Les premières méthodes implémentées ont souvent eu pour objet la résolution du problème classique de TSP, qui constitue un prototype de problème NP-Difficile fréquemment utilisé pour la comparaison de l'efficacité d'une nouvelle technique. Une des méthodes les plus répandue et qui donne de bons résultats est celle dénommée *k*-opt (où $k \geq 2$). Les variantes ayant donné les meilleurs résultats sont celles de 2-opt et 3-opt. Lorsque $k > 3$, l'amélioration peu substantielle des résultats obtenus n'est pas justifiée par l'accroissement notable du temps nécessaire à l'exécution de l'algorithme. Lin et Kernighan ont élaboré en 1973 une autre variante de l'heuristique dénommée Lin-Kernighan, qui est très efficace mais dont la mise en œuvre est beaucoup plus complexe (il est nécessaire d'effectuer plusieurs choix lors de l'élaboration et de l'implémentation de l'algorithme qui ont des répercussions importantes sur son efficacité). Ces trois

variantes sont toujours utilisées aujourd'hui pour la résolution de divers problèmes d'optimisation combinatoire. Elles ont fréquemment été utilisée conjointement avec des méta-heuristiques comme méthode de recherche locale, permettant d'améliorer les solutions obtenues par les heuristiques globales (RS (Cerny, 1985; Kirkpatrick, 1984; Lam J. et Delosme, 1988), AG (Brady, 1985; Martin, Otto et Felten, 1991; Muhlenbein, Gorges-Schleuter et Kramer, 1988) et RT par exemple). Plusieurs études ont cependant noter que l'heuristique globale n'avait pas systématiquement d'influence sur les résultats finaux (Selman). D'après Lin et Kernighan (1973), la qualité de la solution initiale fournie à leur algorithme de recherche locale (qu'elle soit générée aléatoirement ou à l'aide d'une heuristique globale) n'influence pas nécessairement la qualité de la solution finale, démontrant ainsi l'efficacité propre à l'heuristique de recherche locale. Rappelons que ce phénomène a aussi été rencontré par l'algorithme d'André et al. décrit précédemment, effectuant une hybridation des méthodes de PC et de RT, dans lequel la qualité de la solution fournie à l'heuristique de RT n'avait pas forcément une influence sur la qualité de la solution finale. Soulignons que Lin et Kernighan soutiennent que les meilleurs résultats en ce qui concerne leur méthode sont obtenus en exécutant l'algorithme à plusieurs reprises sur des solutions de départ différentes.

Notre étude porte essentiellement sur l'évaluation de l'efficacité de la méta-heuristique d'OCF appliquée à notre problème. Nous n'implémentons par conséquent que les variantes 2-opt et 3-opt de la méthode générique *k*-opt en ce qui concerne l'amélioration locale, la variante de Lin-Kernighan étant relativement complexe à implémenter et nécessitant un ajustement de ses divers paramètres (qui sont généralement fonction du problème) avant de pouvoir atteindre son efficacité maximale. (Il existe par ailleurs de très bonnes implémentations de cet algorithme pouvant être «interfacé» pour la résolution d'un problème défini et dont le code source est immédiatement disponible sous forme de librairie, sans frais pour une utilisation académique.) Les algorithmes 2-opt et 3-opt consistent à appliquer respectivement 2 et 3 mouvements itérativement à une solution donnée afin de tenter d'améliorer la valeur de la fonction-objectif. Supposons une instance du problème de TSP devant être minimisé.

Les algorithmes 2-opt et 3-opt élimineront respectivement 2 et 3 arcs (joignant chacun deux villes) du tour constituant la solution qu'ils cherchent à améliorer, et les remplaceront par 2 et 3 autres arcs respectivement si ces mouvements ont pour conséquence de réduire le coût de la fonction. Les algorithmes prennent fin lorsqu'il n'est plus possible d'améliorer la solution. Lin et Kernighan ont démontré que l'algorithme 3-opt donne des résultats se rapprochant très prés de l'optimum. Les Figures 3.4 et 3.5 démontrent schématiquement le fonctionnement des algorithmes 2-opt et 3-opt.

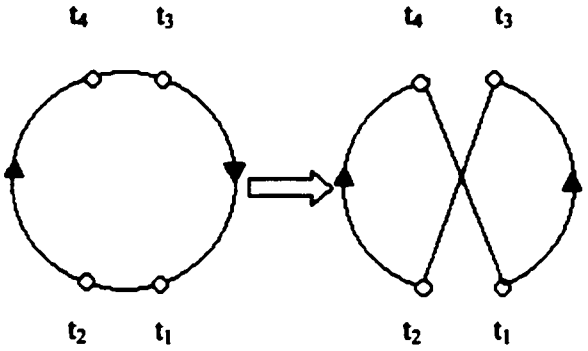

Figure 3.4 Démonstration d'un mouvement 2-opt

Les liens existants entre t_1 et t_2 ainsi que ceux existants entre t_4 et t_3 sont remplacés dans cette Figure 3.4 par t_4 et t_1 d'une part, et par t_3 et t_2 d'autre part. Dans la Figure 3.5, les trois liens x_1, x_2 et x_3 sont supprimés et les liens y_1, y_2 et y_3 feront partie du nouveau tour que l'on suppose plus économique.

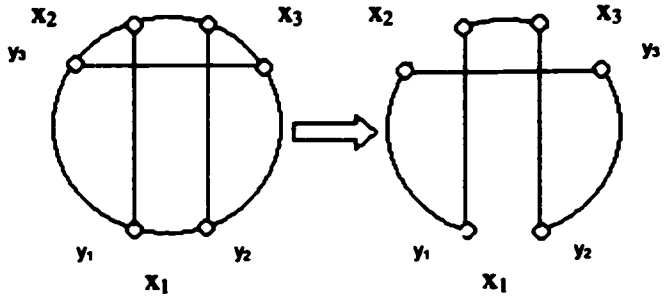

Figure 3.5 Démonstration d'un mouvement 3-opt

L'algorithme générique *k*-opt ne constitue qu'une matrice pouvant être modifiée en fonction du problème (et de son instance) devant être résolu. Le choix de la solution initiale, la nature des mouvements ainsi que les règles relatives à leur sélection ne sont pas dictés par l'algorithme. Il appartient à son utilisateur de l'adapter en fonction de son problème. Ainsi, il est possible avec cette technique d'appliquer plusieurs méthodes en ce qui concerne la sélection des mouvements à effectuer. Deux heuristiques sont largement utilisées. La première consiste à accepter le premier changement rencontré qui améliore la solution actuelle (first-improvement ou hill-climbing). La deuxième technique est celle consistant à rechercher à chaque étape le mouvement local qui améliore le plus la valeur de la fonction (steepest-descent ou greedy local search), la solution obtenue à la fin du cycle étant qualifiée de *k*-optimale. Aussi, afin de contourner les minima locaux, l'algorithme peut être implémenté de manière à momentanément accepter une solution moins économique (ou infaisable), en espérant l'améliorer davantage ultérieurement (ou en espérant rectifier la situation dans le futur) ou à refuser systématiquement une telle solution. Notons enfin que l'algorithme classique de *k*–opt est similaire peu importe le nombre de mouvements effectués (valeur de *k*) à chaque étape, mais que sa complexité augmente parallèlement au nombre de mouvements acceptés ($O(n^k)$).

3.3 Adaptation de la méthode d'OCF à un problème dynamique

Lorsqu'un obstacle survient sur un chemin emprunté par une colonie de fourmis, leur moyen de communication indirect (dépôt de phéromones) sera encore une fois mis en œuvre afin de trouver le chemin le plus court permettant de contourner ce nouvel obstacle (Beckers, Deneubourg et Goss, 1992; Goss, Aron, Deneubourg et Pasteels, 1989). Supposons un chemin initial parcouru par les fourmis menant de leur nid vers la source de nourriture tel qu'illustré à la Figure 3.6 et qu'un obstacle apparaisse sur ce même chemin (Figure 3.7).

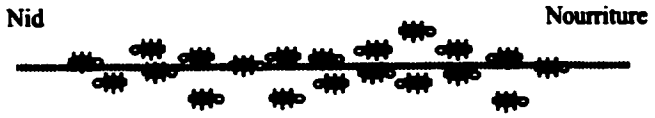

Figure 3.6 Chemin initial

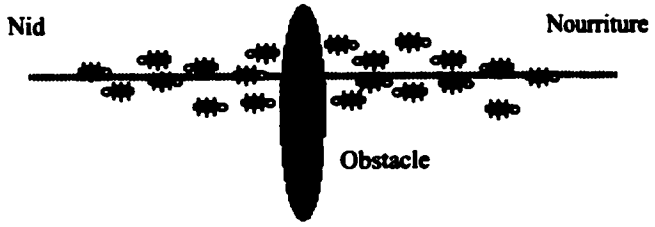

Figure 3.7 Nouvel obstacle

L'environnement ayant été subitement modifié, les fourmis directement positionnées devant l'obstacle le contourneront en choisissant aléatoirement le chemin de droite ou de gauche avec une probabilité d'environ 50 % pour chacun des chemins, comme illustré à la Figure 3.8. Les fourmis ayant par chance choisi le chemin le plus court, rejoindront plus rapidement l'autre partie de la piste initiale coupée par l'obstacle.

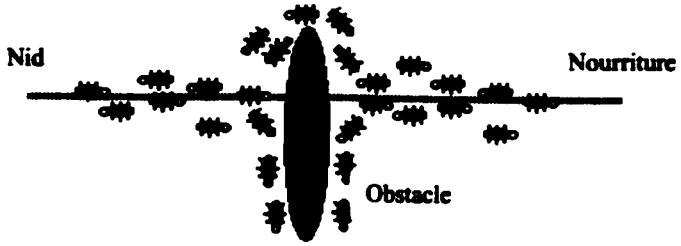

Figure 3.8 Obstacle contourné aléatoirement avant adaptation

Le chemin le plus court recevra donc une quantité de phéromones plus importante par unité de temps, ayant pour conséquence d'attirer davantage de fourmis vers cette alternative. Après un certain délai, l'ensemble des fourmis convergera vers la piste la plus courte contournant l'obstacle, s'adaptant ainsi favorablement à leur nouvel environnement (Figure 3.9).

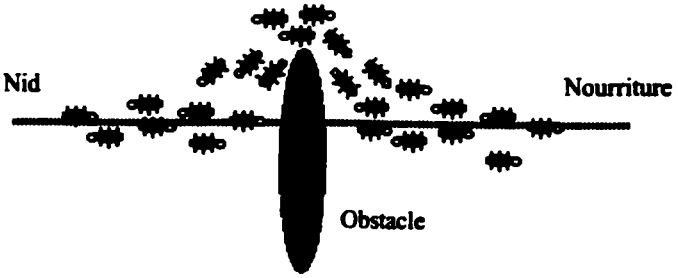

Figure 3.9 Obstacle contourné après adaptation

Tout comme les fourmis biologiques, les fourmis artificielles peuvent s'adapter de manière dynamique à leur environnement. Lorsque le schéma d'appels entre deux cellules change au cours d'une période, la méthode d'OCF permet aux fourmis de se «rendre compte» de ce changement et de s'adapter en conséquence. Le phénomène autocatalytique (positive feedback) induit par les traînées de phéromones met en œuvre un processus d'apprentissage expérimenté par les fourmis.

L'adaptation de notre problème à celui de domiciliation double consiste à rechercher les meilleures solutions que l'algorithme puisse trouver pour l'ensemble des schémas d'appels rencontrés dans un réseau. Après avoir déterminé l'ensemble des liens

physiques à établir, un algorithme dynamique pourrait s'adapter en temps réel aux divers changements intervenant au cours d'une journée. La gestion des appels d'une cellule serait ainsi attribuée au commutateur (parmi les deux éventuels) disposant d'un lien physique avec ladite cellule, qui aurait pour conséquence de réduire le plus possible le nombre de relèves complexes relatives à cette cellule par unité de temps, pour une période donnée.

CHAPITRE IV
IMPLÉMENTATION ET RÉSULTATS

Nous avons procédé à l'implémentation de plusieurs variantes de notre méta-heuristique d'OCF ainsi que de notre technique de recherche locale k-opt, afin de comparer ces différentes alternatives entre elles et de sélectionner celle nous donnant les meilleurs résultats. Le présent chapitre est consacré à la description de ces différentes variantes ainsi qu'à l'analyse de leurs résultats, en comparaison avec les algorithmes décrits au deuxième chapitre. Il est divisé en cinq sections. La première section décrit les caractéristiques de l'implémentation de la méta-heuristique d'OCF. La section deux est consacrée à la description des trois variantes de la technique de recherche locale k-opt ayant été développées. Les résultats relatifs à l'heuristique d'OCF sont analysés à la troisième section alors que l'analyse des résultats des méthodes de recherche locale, hybridées avec la méta-heuristique d'OCF, est effectuée à la section quatre. Enfin, la cinquième section décrit l'implémentation et effectue l'analyse des résultats concernant l'application de nos heuristiques au problème de domiciliatio double.

4.1 Méta-heuristique d'OCF appliquée au problème d'affectation

Il y a plusieurs paramètres qui doivent être ajustés lors de l'implémentation de la méthode d'OCF pour la résolution d'un nouveau type de problème (rappelons que cette méthode n'a jamais été appliquée au problème d'affectation de cellules). L'algorithme est relativement flexible et générique, mais sa structure spécifique doit être adaptée au problème auquel il est appliqué afin d'obtenir les meilleurs résultats possibles. Il est également nécessaire de déterminer la valeur des différents paramètres de l'algorithme donnant les meilleurs résultats. Nous énumérons dans une première sous-section l'ensemble des paramètres qu'il a été nécessaire d'ajuster lors de l'implémentation de l'algorithme. Les choix effectués concernant la structure de leur implémentation sont précisés dans cette sous-section, mais leurs valeurs respectives choisies sont précisées dans la partie consacrée aux résultats. L'algorithme de la méta-heuristique implémentée

pour la résolution de notre problème ainsi que ses différents composants (variables et procédures) sont décrits dans une deuxième sous-section.

4.1.1 Paramètres de la méta-heuristique

La structure utilisée concernant l'implémentation des différents paramètres de notre méthode est décrite ci-après. Nous avons tenu compte, pour arrêter certains choix, des différents résultats obtenus par les algorithmes décrits au deuxième chapitre ainsi que des diverses observations émises par leurs auteurs, afin de pouvoir effectuer une distinction entre les implémentations efficaces concernant certains paramètres et celles donnant de moins bons résultats.

a) Influence relative de l'exploitation des informations du problème et de l'exploration probabiliste de l'espace de recherche

Les fourmis procèdent à la construction d'une solution de manière probabiliste en ajoutant itérativement à leur solution partielle des composants jusqu'à l'obtention d'une solution finale. Elles prennent en compte lors de cette construction, des informations relatives à l'instance du problème solutionné et de leur expérience acquise collectivement en ayant recours aux phéromones. Il leur est avantageux de prendre en compte ces deux éléments afin qu'elles puissent tirer profit de leurs bénéfices respectifs. Lorsqu'une fourmi se retrouve avec une solution partielle quelconque (contenant une série d'affectations de cellules à une série de commutateurs), elle choisit d'intégrer le composant suivant (cellule non encore affectée parmi toutes celles n'ayant pas encore reçu une telle affectation) à sa solution partielle en tenant compte des données du problème (coûts de liaison et de relève pour chaque cellule) ainsi que des traînées de phéromones relatives à la désidérabilité d'effectuer l'affectation considérée dans l'état actuel de la solution partielle. Après avoir déterminé la prochaine cellule à affecter, elle doit ensuite choisir à quel commutateur l'attribuer, toujours en tenant compte des même données du problème et des traînées de phéromones, concernant cette fois la

désidérabilité d'affectation de ladite cellule, relative à chacun des commutateurs disponibles.

La sélection de la prochaine cellule à affecter est effectuée par une fourmi en ayant recours à la règle qualifiée de pseudo aléatoire proportionnelle (*pseudo-random proportional rule*) qui est la suivante :

- choisir avec une probabilité q_0, la cellule pour laquelle le produit entre les informations du problème (η_{ij}^{β}) (pondérées par la valeur de la variable β) et les traînées de phéromones (τ_{ij}^{α}) (pondérées par la valeur de la variable α) relatives à la désidérabilité d'effectuer l'affectation pour chaque cellule à ce moment précis, est le plus élevé : $j = arg\ max_{j \in \mathbf{N}^k_i}\{\tau_{ij}(t)^{\alpha} * \eta_{ij}^{\beta}\})$; (4.1)

- et avec une probabilité $1\text{-}q_0$, sélectionner la cellule en appliquant la probabilité p^k_{ij}, qui correspond à une exploration biaisée :

$$p^k_{ij}(t) = \frac{[\tau_{ij}(t)]^{\alpha} * [\eta_{ij}]^{\beta}}{\Sigma_{l \in \mathbf{N}^k_i}[\tau_{il}(t)]^{\alpha} * [\eta_{il}]^{\beta}} \qquad \text{if } j \in \mathbf{N}^k_i \tag{4.2}$$

Les variables ayant toutes la même signification que celle donnée au chapitre précédent.

La valeur (fractionnaire) sélectionnée du paramètre q_0 donnant les meilleurs résultats est précisée dans la section relative aux résultats. Lorsqu'elle se rapproche de 1, l'algorithme favorise l'exploitation des données connues du problème et des traînées des phéromones alors que lorsqu'elle tend vers 0, l'heuristique met en œuvre une démarche plutôt exploratoire de l'espace de recherche. La valeur attribuée à η_{ij}^{β} relative aux informations du problème est calculée en tenant compte de la règle du moindre regret utilisée et définie au deuxième chapitre par Amoussou et al., et par André et al. dans leurs implémentations de la PC en ce qui concerne l'ordre de sélection des variables (les deux types de coûts, de liaison et de relève, étant utilisés pour effectuer les calculs). Cette règle ayant donnée de bons résultats, nous avons décidé de l'emprunter pour l'implémentation de notre heuristique. Soulignons cependant, qu'elle fait l'objet d'une pondération égale à la quantité de phéromones déposée par les fourmis précédentes ainsi

que d'une pondération déterminée en fonction de la valeur de la variable β. Enfin, la deuxième règle mise en œuvre par André et al. consistant à établir l'ordre desdites cellules à initialiser en ordre croissant du volume d'appels a également été appliquée conjointement avec la première règle. Ainsi, la valeur attribuée à η_{ij}^{β} correspond au ratio du moindre regret divisé par le nombre de communications destiné à la cellule en question (le tout pondéré par la valeur du paramètre β).

La sélection du commutateur auquel attribuer la cellule choisie est effectuée en utilisant des règles semblables à celles définies ci-dessus. La valeur de η_{ij}^{β} est cependant définie par le coût d'attribution d'une cellule à un commutateur donné. Elle est calculée en tenant compte des coûts de liaison et de relève. L'attribution ayant le coût le moins élevé parmi celles possibles disposera de la valeur η_{ij}^{β} la plus élevée (la valeur décroissante de η_{ij}^{β} des autres attributions sera inversement proportionnelle à l'augmentation de leur coût). Concrètement la valeur de η_{ij}^{β} est établie en utilisant le ratio 1/Couts_Totaux.

- une cellule sera ainsi affectée avec une probabilité égale à q_0 au commutateur pour lequel le produit entre les informations du problème (η_{ij}^{β}) (pondérées par la valeur de la variable β) et les traînées de phéromones (τ_{ij}) (pondérées par la valeur de la variable α) relatives à la désidérabilité d'affectation de la cellule en question pour chaque commutateur, est le plus élevé : $j = arg\ max_{j \in \mathbf{N}^k_i}\{\tau_{ij}(t)^{\alpha} * \eta_{ij}^{\beta}\})$;

- et avec une probabilité 1-q_0, en appliquant la probabilité p^k_{ij}, correspondant à l'exploration biaisée telle que définie ci-dessus.

La valeur de la variable q_0 constitue un des paramètres utilisés afin de modifier l'influence relative des informations du problème (données du problème et traînées des phéromones) d'une part et de l'exploration du domaine basée sur une recherche aléatoire d'autre part. Les variables α et β sont de leur côté des paramètres utilisés afin de modifier l'influence relative des informations du problème connues au départ et celles acquises (mémoire collective représentée par les quantités de phéromones) lors de l'exécution de l'algorithme. L'algorithme exploite de manière importante l'expérience acquise par les fourmis lorsque la valeur de α est élevée et lui accorde moins

d'importance dans l'hypothèse inverse. De manière similaire, l'algorithme attribue plus d'influence aux données du problème lorsque la valeur de β est élevée et moins dans le cas contraire.

La règle utilisée en ce qui concerne les quantités de phéromones déposées sur les arcs constitue un autre paramètre modifiable de la méthode nous permettant de modifier l'influence relative des informations acquises (via les phéromones) du problème sur la recherche de solutions. Il existe plusieurs stratégies qui peuvent être implémentées pour le dépôt des phéromones. La stratégie «égalitaire» consiste à accorder à chaque fourmi, le droit de déposer sur les arcs composant sa solution une quantité de phéromones correspondant à la formule suivante :

$$\Delta\tau_{ij}^{k}(t) = \begin{cases} 1/L^{k}(t) & \text{si l'arc } (i,j) \text{ est utilisé par la fourmi } k \\ 0 & \text{sinon} \end{cases} \qquad (4.3)$$

où $L^{k}(t)$ correspond à la solution considérée de la fourmi k. Le droit de déposer des traînées de phéromones peut au contraire, n'être accordé qu'à la fourmi ayant obtenu la meilleure solution. Il est également possible d'attribuer ce droit à l'ensemble des fourmis mais en permettant à la fourmi disposant de cette meilleure solution de déposer davantage de phéromones. Ces stratégies «élitistes» peuvent être implémentées en accordant un de ces privilèges à la fourmi ayant obtenu la meilleure solution soit d'un cycle, soit depuis le début de l'exécution de l'algorithme. Il est enfin possible d'accorder aux fourmis un droit de dépôt qui est proportionnel à la qualité de leur solution.

La flexibilité de notre implémentation permet de mettre en œuvre trois de ces stratégies. La quantité de phéromones pouvant être déposée par l'ensemble des fourmis et celle pouvant être déposée uniquement par la fourmi ayant obtenu la meilleure solution sont contenues dans deux variables distinctes. Il sera ainsi possible d'utiliser une stratégie purement égalitaire en accordant une valeur positive à la variable chargée de contenir la quantité de phéromones pouvant être déposée par l'ensemble des fourmis et une valeur nulle à la variable contenant la quantité de phéromones pouvant être déposée par la fourmi ayant obtenu la meilleure solution. La variante intermédiaire,

accordant à chaque fourmi le droit de déposer des phéromones sur les composants de sa solution lorsqu'elle dispose d'une solution achevée et accordant le privilège à la fourmi ayant obtenue la meilleure solution lors d'un cycle de déposer davantage de phéromones sur les composants de sa solution, peut être mise en œuvre en attribuant une valeur positive aux deux variables. Ces stratégies permettent aux fourmis de s'influencer mutuellement de manière dynamique lorsqu'elles procèdent à la construction de leurs solutions. Le poids accordé à chaque variable déterminera le caractère plutôt égalitaire ou plutôt élitiste de la stratégie mise en œuvre. Enfin, il est possible de recourir à une stratégie purement élitiste en attribuant une valeur positive uniquement à la variable contenant la quantité de phéromones pouvant être déposée par la fourmi ayant obtenu la meilleure solution lors d'un cycle, la valeur de la variable concernant le droit de dépôt accordé à l'ensemble des fourmis restant nulle.

b) Moment de la mise à jour des traînées de phéromones

Le moment de la mise à jour des traînées de phéromones constitue un deuxième facteur pouvant avoir une influence non négligeable sur l'efficacité de l'algorithme. Il existe trois types de mise à jour. Les deux premières techniques sont effectuées par les fourmis. Elles peuvent procéder à la mise à jour soit à toutes les fois qu'elles ajoutent un composant à leur solution (*online step-by-step update*), soit après avoir trouvé leur solution finale (*online delayed update*). Les fourmis s'influencent mutuellement en parallèle lors de la construction de leurs solutions partielles, lorsque la mise à jour est effectuée à chaque étape de la construction. Les mises à jour n'influencent que les fourmis suivantes lorsqu'elles sont retardées jusqu'à l'accomplissement des solutions. Notre implémentation utilise cette deuxième technique pour trois raisons. La première technique est d'abord très exigeante en terme de temps d'exécution, puisqu'il est nécessaire de procéder à un dépôt des phéromones à chaque fois qu'une seule fourmi intègre un nouveau composant à sa solution partielle, si l'on désire que l'état des différents composants soit connu en «temps réel». Il nous est par ailleurs nécessaire de connaître la qualité des solutions finales avant de procéder à la mise à jour des traînées,

si nous désirons mettre en œuvre de manière cohérente une variante élitiste. Enfin, notre implémentation n'est que partiellement parallèle. L'ensemble des fourmis actives lors d'un cycle procède à la construction de leur solution de manière séquentielle. Ce choix à été effectué en raison de la possibilité d'appliquer une variante élitiste à la résolution d'un problème. Il n'eût pas été cohérent d'effectuer une mise à jour des traînées en parallèle lors de la résolution d'un problème à l'aide d'une stratégie élitiste. Il est en effet nécessaire de connaître la qualité des solutions finales avant d'effectuer la mise à jour des traînées lorsqu'une telle stratégie est appliquée. Ainsi, dans la mesure où la mise en œuvre d'une stratégie élitiste ne nous permet pas d'effectuer une mise à jour des traînés en parallèle, il n'était pas justifié d'accroître la complexité de l'algorithme et d'augmenter son temps d'exécution en implémentant une version purement parallèle de la méthode. Notre implémentation est cependant partiellement parallèle puisque chaque fourmi met à jour ses traînés (dans le cas d'une implémentation purement ou «en partie» égalitaire) lorsqu'elle achève sa solution, influençant ainsi ses consœurs au cours du même cycle et pour les cycles futurs.

Nous avons par ailleurs, utilisé le troisième type de mise à jour qui est effectué, non pas par les fourmis, mais à l'aide d'un démon (*deamon*). Cette mise à jour consiste à déposer des phéromones sur les composants inclus dans la meilleure solution trouvée lors d'un cycle. Les fourmis ne disposant pas d'une mémoire leur permettant de se souvenir de la meilleure solution depuis le départ, il appartient au démon de comparer les diverses solutions du présent cycle afin de sélectionner la meilleure parmi l'ensemble. La quantité de phéromones déposée relative à cette meilleure solution globale est ajustée jusqu'à l'obtention des meilleurs résultats.

c) Évaporation des phéromones

Il est possible d'appliquer un coefficient d'évaporation aux traînées de phéromones afin de permettre aux fourmis de ne plus emprunter certains arcs ne menant pas à de bonnes solutions ou pour assurer une exploration importante de l'espace de recherche. Nous n'effectuons pas d'évaporation pendant l'exécution d'une boucle

(constituée de plusieurs cycles), mais uniquement lors de leur «initialisation». (L'exécution intégrale de l'algorithme consiste en plusieurs boucles, elles-mêmes composées de plusieurs cycles, tel qu'il est expliqué ci-après.) L'exécution de plusieurs boucles permet de diversifier la recherche vers différentes zones du domaine. Chaque boucle de l'algorithme démarre une nouvelle recherche en éliminant toutes les traînées déposées lors de la boucle précédente. Cette évaporation intégrale appliquée à chaque initialisation garantit davantage une diversification de la recherche. L'évaporation partielle effectuée pendant l'exécution d'une boucle aurait par ailleurs un effet contraire à la philosophie élitiste pouvant être mise en œuvre. Aussi, le type de mise à jour des phéromones a un effet qui peut être assimilé au phénomène d'évaporation, sans en apporter ses contradictions. Lorsqu'une cellule est rarement affectée à un commutateur par exemple, la petite quantité de phéromones qui aurait éventuellement été déposée antérieurement sur la représentation de cette affectation, sera de moins en moins prise en compte par les fourmis lors de la construction de leurs solutions, qui constateront une quantité relativement plus importante sur les affectations alternatives. Soulignons enfin, que cette évaporation est difficile à mettre en œuvre de manière efficace.

d) Nombre et mémoire des fourmis

Le paramètre concernant le nombre de fourmis à utiliser afin d'obtenir les meilleurs résultats sera fixé expérimentalement, comme l'ensemble des paramètres de notre algorithme. Il a été démontré (Dorigo et Stützle) que l'utilisation de $m > 1$ fourmis pour l'implémentation d'une heuristique ACO permettait à l'algorithme de converger plus rapidement vers de bonnes solutions. Il serait en effet imaginable d'effectuer un nombre d'itérations de l'algorithme supérieur avec une seule fourmi qu'avec plusieurs, donnant une quantité de solutions identiques dans les deux cas (soit $Nb\_Solutions = 1*Nb\_iterations\_UneFourmi = m*Nb\_iterations\_PlusieursFourmis$, où m est le nombre de fourmis et $Nb\_iterations\_UneFourmi/m = Nb\_iterations\_PlusieursFourmis$). La fourmi effectuant une recherche séquentielle à chaque itération, ne bénéficierait pas dans cette hypothèse de l'expérience acquise par ses consœurs. Plus théoriquement, une

procédure consistant à construire de manière séquentielle une seule solution à partir d'un état du système (quantité de phéromones disponible sur chaque affectation), n'est pas équivalente à une procédure consistant à construire de manière parallèle plusieurs solutions à partir de ce même état initial. La construction de plusieurs solutions effectuée en parallèle a une influence non comparable sur l'état du système utilisé par le cycle suivant, par rapport à la construction de ces solutions effectuée de manière séquentielle (même si le nombre de solutions générées est identique). Il ne serait par ailleurs pas possible d'implémenter une stratégie élitiste avec une seule fourmi. Notons enfin, que la mémoire individuelle de l'unique fourmi se confondrait dans ce cas avec la mémoire «collective» de la «colonie». Notre implémentation fournit à chaque fourmi une mémoire individuelle, lui permettant de retracer son chemin afin d'y déposer des phéromones et d'une mémoire collective permettant à un démon de déposer davantage de phéromones sur le chemin de la fourmi ayant obtenu la meilleure solution depuis le début d'un cycle.

Bien que notre implémentation ne soit que partiellement parallèle en raison des justifications données préalablement, elle bénéficie néanmoins globalement du parallélisme propre à l'utilisation de plusieurs fourmis (à condition d'en utiliser plusieurs bien entendu, notre implémentation offrant la possibilité d'en n'utiliser qu'une seule) lorsqu'une stratégie élitiste est utilisée. L'exécution séquentielle de la construction des solutions ne se limite qu'à la procédure interne d'un cycle, l'algorithme étant composé de plusieurs cycles, eux-mêmes réitérés à plusieurs reprises au sein de plusieurs boucles. Ainsi, les fourmis démarreront leur recherche lors d'un nouveau cycle avec un état du système qui est différent de celui précédent, modifié par les phéromones qui y auront été déposées lors du cycle antérieur. Les phéromones déposées par l'ensemble des fourmis lors des cycles précédents et notamment par les fourmis ayant obtenu la meilleure solution au cours de ces cycles dans le cas d'une stratégie élitiste, influencent ainsi l'ensemble des fourmis actives au cours du présent cycle. Globalement, les fourmis bénéficient donc de l'expérience acquise par leurs consœurs par le biais des dépôts de phéromones effectués par elles-mêmes et par le démon.

e) Voisinage des fourmis et respect des contraintes

Le voisinage d'une fourmi est constitué de l'ensemble des cellules n'ayant pas encore été affectées en ce qui concerne la sélection de la prochaine cellule à considérer. Son voisinage est constitué de l'ensemble des commutateurs disposant des ressources suffisantes en ce qui concerne l'affectation de la cellule sélectionnée. Chaque solution partielle ou finale des fourmis doit respecter les contraintes d'unicité en ce qui concerne l'affectation unique. Notre implémentation donne la possibilité à l'utilisateur de choisir si l'algorithme d'OCF doit systématiquement respecter les contraintes de capacité des commutateurs ou s'il est temporairement possible de les violer au cours de la construction d'une solution. Toutes les solutions finales doivent cependant respecter les contraintes de capacité.

f) Optimisations globale et locale

L'heuristique d'OCF tente dans un premier temps de distinguer de manière globale les zones contenant des minimums locaux de celles menant vers des sommets. Il lui appartient ensuite de diriger une recherche plus intensive à l'intérieur des zones jugées favorables soit par elle-même, soit en ayant recours à l'HOL (Heuristique d'optimisation Locale, k-opt) afin de procéder à cette recherche. Notre algorithme procède à sa démarche de recherches globale et locale en équilibrant deux approches. Il effectue une intensification vers les zones perçues comme susceptibles de contenir de bonnes solutions en exploitant les informations du problème, tout en diversifiant l'exploration de l'espace de recherche afin de couvrir le plus largement possible le domaine. L'algorithme à recours à différents mécanismes et paramètres afin de mettre en œuvre cet équilibre entre intensification et diversification de la recherche. Le premier de ces mécanismes, vu précédemment, consiste à effectuer une recherche biaisée du domaine, en ayant recours à la règle pseudo aléatoire proportionnelle. Cette règle lui permet d'intensifier graduellement sa recherche vers les zones jugées favorables. La probabilité accordée à $1\text{-}q_0$ lui permet de s'assurer de ne pas converger trop brusquement vers un optimum local et de diversifier sa recherche. Seule l'expérimentation nous permettra de

déterminer la valeur optimale à attribuer à la variable q_0, permettant à l'algorithme de trouver l'équilibre approprié entre exploitation des données et exploration du domaine, menant aux meilleures solutions. L'ajustement des paramètres α et β concourt également à atteindre cet équilibre.

Le deuxième mécanisme implémenté afin d'équilibrer la recherche entre une intensification et une diversification de l'espace de recherche est le recours à ce que nous dénommons les cycles et les boucles. L'algorithme effectue en effet plusieurs cycles et plusieurs boucles, dont les nombres sont fonction de la valeur des paramètres fournie par l'usager. Les cycles permettent à l'algorithme d'intensifier sa recherche vers certaines zones alors que les boucles lui permettent au contraire de diversifier sa recherche vers de nouvelles zones. La valeur fournie par l'utilisateur servant à déterminer le nombre de cycles correspond au nombre d'itérations successives que l'algorithme devra effectuer sans rencontrer d'amélioration avant de pouvoir procéder à une ré-initialisation des tableaux contenant les quantités de phéromones. Ainsi, si la valeur de la variable *NCycles* est égale à 2, l'algorithme continuera sa recherche dans une zone locale (sans ré-initialiser les tableaux de phéromones) jusqu'à ce qu'il ne soit pas parvenu à améliorer la valeur de sa meilleure solution actuelle pendant 2 cycles successifs. À chaque fois qu'une solution moins coûteuse est trouvée, elle remplace la meilleure solution actuelle et la valeur de la variable *NcyclesSsAmel* (Nombre de Cycles Sans Amélioration) est ré-initialisée à 0. L'algorithme intensifie ainsi graduellement à l'aide de ce mécanisme ses recherches à chaque cycle dans certaines zones locales, lui permettant de converger éventuellement vers de bonnes solutions.

Afin de s'assurer de couvrir substantiellement l'espace de recherche, l'algorithme ré-initialise au contraire à chaque boucle son domaine de recherche en éliminant l'intégralité des phéromones déposées antérieurement et en dirigeant la recherche vers d'autres zones au départ. L'état de départ du système se définit de la manière suivante. Chaque fourmi démarre une recherche en disposant dans sa solution initiale partielle que d'une cellule affectée au commutateur du moindre coût. Elle procède ensuite à la construction de sa solution à partir de cet état initial. L'algorithme effectue une

diversification des états de départ en attribuant à chacune des fourmis une cellule n'ayant jamais encore été affectée lors d'une initialisation au cours de l'ensemble des boucles. Afin de ne pas affecter de manière séquentielle chaque cellule aux fourmis, un coefficient est appliqué à chaque sélection en parcourant la liste des cellules existantes. Ce coefficient dépend du nombre de fourmis et du nombre de cellules. Il est déterminé en divisant le nombre de cellules par le nombre de fourmis utilisées, arrondi à l'entier inférieur le plus rapproché (*Arr_Inf(Nb_Cellules/Nb_Fourmis)*).

Supposons par exemple, que le nombre de cellules soit de 100 et le nombre de fourmis soit égal à 7. Le coefficient sera de 14 (100/7 = 14.285). Ainsi, lors de l'initialisation de la première boucle, la première cellule sera attribuée à la première fourmi de la colonie. La deuxième fourmi se verra attribuer la 1 + 14 = 15ème cellule du réseau. La 15 + 14 = 29ème cellule sera attribuée à la troisième fourmi, alors que les fourmis suivantes se verront attribuer une cellule d'une manière similaire. Lors de l'initialisation de la seconde boucle, la deuxième cellule sera attribuée à la première fourmi alors que la deuxième fourmi de la colonie recevra la 16ème cellule comme solution partielle de départ. Les attributions aux fourmis lors de l'initialisation des boucles successives suivant une logique similaire. Nous avons recours à ce coefficient puisque la numérotation des cellules est fréquemment effectuée en considérant itérativement chaque cellule se trouvant non éloignée géographiquement de la cellule précédemment numérotée. En appliquant un coefficient à la sélection des cellules, l'heuristique démarre chacune de ses recherches avec des solutions partielles ayant moins de risque de contenir des cellules contiguës. La probabilité pour une fourmi de démarrer la construction de sa solution dans la même région qu'une de ses consœurs, s'en trouve réduite par l'application de ce coefficient. Lorsque l'algorithme aura terminé sa 14ème boucle, les 98 première cellules auront toutes été affectées lors d'une initialisation. Seules les cellules restantes (2 dans notre exemple) ne seront jamais attribuées à une fourmi au démarrage d'une boucle. Cette exception n'entraîne théoriquement aucune conséquence puisque nous avons implémenté ce mécanisme d'initialisation des boucles, uniquement afin de couvrir le plus largement possible le

domaine de recherche, tout en maintenant le nombre de boucles le plus petit possible afin de ne pas augmenter le temps d'exécution. Notre algorithme bouclera donc à 14 reprises dans notre exemple, le nombre de boucles étant fonction du nombre de fourmi. Si l'utilisateur désire accorder à l'algorithme l'autorisation d'effectuer un nombre déterminé de boucles qui soit inférieur à celui calculé au moyen du ratio *Nb_Cellules/Nb_Fourmis*, il doit fournir une valeur positive correspondant au nombre de boucles désirées. L'algorithme procédera aux initialisations de la même manière que décrite ci-dessus, mais effectuera un nombre plus réduit d'initialisations. La valeur 0 fournie par l'utilisateur indique à l'heuristique d'effectuer un nombre de boucles correspondant au ratio *Nb_Cellules/Nb_Fourmis*.

Le nombre de boucles et de cycles donnant les meilleurs résultats en termes de coûts et de temps d'exécution sera déterminé expérimentalement. L'implémentation spécifique de notre algorithme nous permet d'ajuster la valeur de l'ensemble des paramètres traités dans la présente sous-section jusqu'à l'obtention des meilleurs résultats.

4.1.2 Détails d'implémentation et algorithme

Les détails d'implémentation de la méthode et la description de l'algorithme ainsi que de ses différentes variantes sont donnés ci-après dans la présente sous-section.

a) Détails d'implémentation

L'algorithme a été développé en langage C++ à l'aide de l'éditeur Visual C++. L'heuristique elle-même ne comprend que du C++ standard. Il est donc possible de le porter et de le compiler sur n'importe quelle plate-forme acceptant ce standard. L'environnement d'exécution utilise cependant quelques objets de la classe MFC (*Microsoft Fundation Class*) concernant le système de fenêtrage. L'heuristique comportant plusieurs paramètres modifiables, nous avons voulu fournir à l'utilisateur un système de fenêtres (tout en minimisant son délai d'implémentation) afin de lui permettre de modifier rapidement la valeur de ces paramètres sans devoir recourir à

l'ouverture d'un fichier texte externe. Notons qu'il est possible de porter l'ensemble de ces objets de la classe MFC vers l'ensemble des variantes Unix les plus utilisées ainsi que vers la plupart des architectures courantes (Sun, DEC, PowerPc...). L'environnement d'exécution est WIN2000-32 Bits sur architecture Intel. La vitesse du processeur est de 350 MHz, disposant de 96 Mega octets de mémoire vive. Les données utilisées afin d'effectuer les tests sont les mêmes que celles utilisées par André et al. (2001). L'ensemble des paramètres devant être fournis par l'utilisateur (via un fichier pouvant être directement ouvert dans l'environnement d'exécution ou au moyen d'un éditeur de texte) figure dans le tableau suivant.

Tableau 4.1 Paramètres devant être fournis par l'utilisateur

Paramètre	Fonction
Ncell	Nombre de cellules de l'instance
Ncom	Nombre de commutateurs de l'instance
NFourmis	Nombre de fourmis utilisées (7 étant le nombre maximal)
NCycles	Nombre de cycles autorisés
Parq0Sel	Paramètre *q0* relatif à la sélection
Parq0Att	Paramètre *q0* relatif à l'affectation (attribution)
ParAlphaSel	Paramètre α concernant la sélection de la prochaine cellule à affecter
ParBetaSel	Ce paramètre n'est pas fourni par l'utilisateur mais calculé en soustrayant ParAlphaSel de 1
ParAlphaAtt	Paramètre α concernant l'affectation
ParBetaAtt	Ce paramètre n'est pas fourni par l'utilisateur mais calculé en soustrayant ParAlphaAtt de 1
IndPrB	Indicateur concernant le nombre de boucles à effectuer. Égale à 0 lorsque l'utilisateur désire qu'il soit égal au ratio Nb_Cellules/Nb_Fourmis et > 0 lorsqu<il désire précisé un nombre inférieur
QtePhe	Quantité de phéromones déposée par l'ensemble des fourmis
Qtegb	Quantité de phéromones déposée par la fourmi ayant obtenu la meilleure solution d'un cycle
Typeaff	Type d'affectation (soit Moindre Coût, soit Moindre Augmentation comme expliqué dans la prochaine section)
Cappec	Égale à 1 si la capacité des commutateurs doit être respectée de manière systématique, et égale à 0 dans le cas contraire
Fkopt	Égale à 0 si aucune procédure d'optimisation locale n'est mise en œuvre, et égale à 1, 2 ou 3 lorsqu'une de ces trois procédures est mise en œuvre (ces procédures étant décrites ci-après)

b) Algorithme de la méta-heuristique

Le pseudo code de la Figure 4.1 décrit de manière simplifiée l'algorithme de la méta-heuristique implémentée.

```
ProcMetaheuristiqueOCF
        Initialiser Algorithme
        Tant Que NbouclesExec <= NbouclesMax Éffectuer
                Initialiser Boucle
                Tant Que NcyclesExec <= NcyclesSansAmel Éffectuer
                        Initialiser Cycles
                        Tant que NFourmisExec <= NFourmisMax Effectuer
                                Initialiser Fourmis
                                Tant Que NCellAff <= NCellTotal Effectuer
                                        Sélectionner Prochaine Cellule à Affecter (Règle Pseudo Aléatoire)
                                        Effectuer Affectation (Règle Pseudo Aléatoire)
                                        Effectuer MAJ Appropriées (Capacité Commutateurs, Coûts...)
                                Fin Tant Que
                                Appeler k-opt
                                Mettre a Jour Traînées de Phéromones
                        Fin Tant Que
                        Évaluer Qualité de la Solution
                        Retenir Meilleure Solution parmi l'Ensemble des Fourmis
                        Mettre à Jour Traînées de Phéromones Concernant Meilleure Solution
                Fin Tant Que
                Comparer Meilleure Solution Actuelle avec Meilleure Solution Précédente
                Retenir Solution Actuelle si Meilleure que Précédente
        Fin Tant Que
        Sélectionner Meilleure Solution Globale
Fin ProcMetaheuristiqueOCF
```

Figure 4.1 Pseudo-code simplifié de la méta-heuristique

L'algorithme applique la règle pseudo aléatoire proportionnelle décrite ci-dessus afin d'effectuer la sélection de la prochaine cellule à affecter ainsi que le choix de son commutateur. Il utilise la valeur des paramètres *Parq0Sel* (pour la sélection de la prochaine cellule à affecter) et *Parq0Att* (pour déterminer le commutateur auquel affecter la cellule choisie) fournie par l'utilisateur afin de mettre en œuvre la règle. Il génère en premier lieu une première valeur aléatoire comprise entre 1 et 100. Lorsque cette valeur est inférieure à la valeur désirée par l'utilisateur (*Parq0Sel*) en ce qui concerne la sélection de la prochaine cellule à affecter, l'algorithme utilise le premier

argument de la règle afin d'effectuer son choix. Dans le cas contraire, il met en œuvre le second argument de la même règle consistant en une sélection aléatoire. Nous avons apporté une modification au second argument de la règle tel que décrit ci-dessus afin de le distinguer du premier. Dans la mesure où la valeur de notre variable η_{ij}^{β} correspondant aux données du problème est déterminée en fonction des *moindres regrets* calculés sur l'ensemble des cellules n'ayant pas encore été affectées, l'application du second argument tel que défini par la théorie aurait eu un effet identique au premier argument en ce qui concerne la sélection de la prochaine cellule à affecter. L'algorithme génère au lieu une seconde valeur aléatoire afin de déterminer la prochaine cellule à affecter. La cellule correspondant à la valeur générée est sélectionnée (celle s'en rapprochant le plus lorsque la cellule correspondant à la valeur aléatoire dispose déjà d'une affectation). Après avoir déterminé la prochaine cellule à affecter, l'algorithme génère une autre variable aléatoire afin de déterminer quel argument de la règle à appliquer pour choisir l'affectation. Il procède de manière similaire en comparant la valeur générée avec la valeur du paramètre *Parq0Att* afin d'orienter sa procédure d'affectation. Lorsque le premier argument de la règle est appliqué, le commutateur ayant la valeur η_{ij}^{β} la plus élevée et disposant des ressources suffisantes est sélectionné. L'algorithme génère une quatrième valeur aléatoire dans le cas où le deuxième argument doit être mis en œuvre. La cellule est dans ce cas affectée au commutateur correspondant à la valeur générée s'il dispose des ressources suffisantes (celui s'en rapprochant le plus lorsque le commutateur en question ne dispose pas des ressources nécessaires).

L'algorithme offre le choix à l'utilisateur entre deux alternatives en ce qui concerne l'affectation des cellules aux commutateurs du réseau. La première alternative est celle consistant à appliquer la règle du *moindre coût* en tenant compte des coûts de liaison et de relèves (relatives aux cellules ayant déjà été affectées) afin de déterminer les valeurs de la variable η_{ij}^{β} correspondant aux données du problème. Rappelons que notre choix s'effectue (ou plutôt peut s'effectuer) en tenant compte non seulement de la valeur de la variable η_{ij} mais également des traînées de phéromones et des paramètres α et β qui ont pour objet de pondérer ces valeurs. Il est cependant possible d'attribuer une

valeur «nulle» à l'ensemble de ces éléments afin d'obtenir une démarche ne tenant compte que de la valeur de cette variable η_{ij}. La seconde alternative, que nous dénommons *moindre augmentation*, consiste à déterminer les valeurs de la variable η_{ij} en calculant l'augmentation anticipée des coûts (liaison et relèves – en tenant compte des affectations précédentes) qu'aurait chaque affectation possible. Les valeurs de la variable η_{ij} sont calculées dans cette alternative en calculant dans un premier temps la différence entre les coûts totaux (de liaison et de relèves) de chacun des commutateurs disposant de la capacité suffisante relative à l'affectation d'une cellule déterminée et le commutateur de moindres coûts concernant cette même cellule. L'algorithme ajoute ensuite à cette différence, les coûts de relèves relatifs à l'ensemble des cellules affectées. Le commutateur pour lequel la somme de ces deux valeurs (différence et relèves) est la moins élevée est sélectionné pour effectuer l'affectation. Cette démarche a pour effet d'attribuer un poids légèrement plus important aux coûts de relèves par rapport aux coûts de liaison puisqu'ils sont pris en compte à deux reprises lors du calcul des valeurs de la variable η_{ij}. Les commutateurs disposant de coûts de relèves moins importants (relatifs à une cellule déterminée) seront avantagés avec cette technique. Nous avons implémenté cette stratégie afin d'observer ses résultats lorsqu'elle est appliquée à des données ayant des coûts de relèves relativement importants par rapport aux coûts de liaisons. Notons que cette alternative utilise l'ensemble des paramètres précédemment définis, à l'exception des coefficients α et β concernant l'affectation. Ces coefficients peuvent cependant être utilisés en ce qui concerne la sélection des cellules à affecter. Cette implémentation simplifiée n'avait pour objectif que d'évaluer les résultats d'une démarche différente. L'alternative exécutée dépend de la valeur du paramètre typeaff définie par l'utilisateur. La valeur 1, dirige l'algorithme vers l'affectation basée sur le *moindre coût*, alors que la valeur 2 le dirige vers l'affectation tenant compte de la *moindre augmentation*.

Notons enfin, que l'utilisateur peut indiquer à l'algorithme au moyen de la variable Cappec s'il doit systématiquement respecter la capacité des commutateurs ou s'il lui est possible de la violer temporairement, en ayant soin de la rétablir par la suite. La valeur 1

lui indique qu'il doit s'assurer de respecter l'ensemble des capacités à toutes les étapes de sa démarche alors que la valeur 0 lui donne la possibilité de les violer temporairement. Lorsque l'utilisateur décide de n'autoriser aucune violation (même temporaire), il pourra exécuter l'heuristique d'OCF seule, sans avoir recours à une variante k-opt ou, en ayant recours à la première de ces variantes (expliquées ci-après). Lorsqu'il admet une violation temporaire des capacités, il devra recourir soit à la deuxième, soit à la troisième variante k-opt. Soulignons que la technique de la moindre augmentation concernant la sélection de l'affectation doit être utilisée conjointement avec la variante n'admettant aucune violation des capacités. La technique de moindre coût peut être utilisée avec l'une ou l'autre des variantes.

Après avoir affecté l'ensemble des cellules aux commutateurs du réseau et suite à l'exécution éventuelle de la méthode k-opt afin de tenter d'améliorer (ou de rendre admissible) sa solution initiale, la fourmi venant d'achever sa solution met à jour ses traînés de phéromones (autant en ce qui concerne l'ordre de sélection des cellules que pour leur affectation). Ces traînées auront pour effet d'influencer les choix effectués par les fourmis suivantes. Lorsque l'ensemble des fourmis dispose d'une solution complète, l'algorithme conserve la meilleure d'entre elles. Il met éventuellement à jour les traînés relatives à cette meilleure solution (dans l'hypothèse d'une stratégie élitiste) avant d'évaluer s'il doit exécuter un autre cycle. Il procède à l'exécution d'un autre cycle lorsque la meilleure solution venant d'être générée est inférieure à la meilleure du cycle précédent (auquel cas la meilleure solution du cycle précédent est substituée par la meilleure solution actuelle) ou lorsque l'algorithme n'a pas effectué un nombre de cycles sans amélioration, supérieur au nombre *NcyclesSsAmel* autorisé (la valeur de cette variable est incrémentée de 1 lorsque l'algorithme n'est pas parvenu à améliorer la solution lors du cycle considéré et est réinitialisée à 0 dans le cas contraire). Enfin, l'algorithme retourne la meilleure solution globale parmi les meilleures de l'ensemble des boucles effectuées.

4.2 Heuristique de recherche locale *k*-opt

Nous avons implémenté trois variantes de l'heuristique k-opt. La variante exécutée dépend de la valeur de la variable *fkopt* fournie par l'utilisateur. La valeur 1, qui doit être utilisée conjointement avec la valeur 1 pour la variable Cappec concernant le respect des capacités, dirige l'algorithme vers la première variante qui a pour objet de tenter d'améliorer la solution fournie par l'heuristique d'OCF. Les deuxièmes et troisièmes variantes (exécutées lorsque la valeur de *fkopt* est égale à 1 ou à 2 respectivement) ont pour objet de rétablir la capacité des commutateurs en utilisant une démarche similaire à la méthode k-opt classique. Elles sont donc utilisées lorsqu'il aura été accordé à l'heuristique d'OCF le droit de temporairement violer la capacité des commutateurs. La valeur 0 attribuée à la variable fkopt indique à l'algorithme d'exécuter une recherche uniquement à l'aide de la méthode OCF.

4.2.1 Première variante k-opt

L'heuristique locale démarre sa procédure en calculant pour chaque cellule n'ayant pas été affectée au commutateur optimal (celui disposant des coûts les moins élevés pour la cellule en question) en raison du manque de ressource, la différence entre les coûts (de liaison et de relèves) relatifs au commutateur auquel elle est présentement affectée et ceux relatifs au commutateur optimal. Elle trie ensuite lesdites cellules en ordre décroissant en fonction de l'importance de la différence des coûts. Elle tente itérativement de ré-affecter chaque cellule n'ayant pas été attribuée au commutateur optimal en suivant l'ordre du tri, de la manière suivante.

Elle procède à un transfert simple de la cellule concernée vers son commutateur optimal lorsqu'il dispose de la capacité suffisante. Il se peut en effet qu'un commutateur puisse disposer à présent des ressources suffisantes afin de gérer ladite cellule en raison des ré-affectations effectuées lors des itérations précédentes au moyen des mouvements décrits ci-après. Lorsqu'il n'est pas possible d'effectuer un tel transfert, l'algorithme tente de procéder à un échange entre la cellule actuellement traitée et une tierce cellule convoitant chacune le commutateur auquel elles sont respectivement affectées. Cet

échange peut s'effectuer uniquement s'il n'a pas pour effet de violer la capacité d'un des commutateurs en question. S'il n'a toujours pas été possible de ré-affecter ladite cellule, l'algorithme procède à une double modification de la cellule concernée et d'une cellule tierce désirant aussi être ré-affectée. La cellule tierce n'ayant pas dans ce cas pour commutateur optimal celui de sa consœur. Ces modifications ne pourront intervenir, encore une fois, que si elles n'ont pas pour effet de violer la capacité d'un des commutateurs. Enfin, s'il n'est toujours pas parvenu à ré-affecter la cellule souhaitant se faire ré-affecter, l'algorithme procède à une double modification de ladite cellule et d'une cellule tierce ne désirant pas une ré-affectation, mais dont les modifications ont pour effet de diminuer l'ensemble des coûts du réseau. Plus concrètement, la diminution des coûts procurée par la ré-affectation de la cellule concernée, doit être supérieure à l'augmentation des coûts provoquée par la ré-affectation de la cellule ne souhaitant pas se faire ré-affectée, étant déjà attribuée à son commutateur optimal. La capacité des commutateurs doit être respectée afin de pouvoir procéder à cette double modification. La procédure s'arrête lorsque l'algorithme a tenté à une reprise, en essayant l'ensemble des mouvements décrits ci-dessus, de ré-affecter chacune des cellules souhaitant être ré-affectée. L'algorithme de cette première variante est décrit ci-après.

```
Proc_k-opt
        Calculer Différence entre les Coûts de la Cellule Actuelle et ceux de la Cellule Optimale
        Trier les Cellules en Ordre Décroissant en Fonction de la Différence de Coûts
        Tant Que Nitération <= NCelluleDevantEtreReaffectées
                Si Commutateur Convoité Dispose de Capacité
                        Effectuer Transfert Simple
                Sinon, Si 2 Modifs Exigées et si Com Disposeront Capacité suite à l'Échange
                        Effectuer Échange
                Sinon, Si 2 Modifs Exigées et si Com Disposeront Capacité suite aux Modifs
                        Effectuer Double Modif
                Sinon, Si 1 Seule Modif Exigée, si Com Disp. Cap. et si les Coûts Totaux Diminueront
                        Effectuer Double Modif
        Fin Tant Que
Fin Proc_k-opt
```

Figure 4.2 Pseudo-code simplifié de la première variante k-opt

4.2.2 Deuxième variante k-opt

Cette deuxième variante a pour effet de rétablir la capacité des commutateurs lorsque leur violation a été autorisée lors de l'exécution de l'heuristique d'OCF. L'algorithme débute cette variante en dénombrant le nombre de commutateurs dont la capacité n'a pas été respectée. Une fois ce dénombrement effectué, il les trie en ordre croissant en fonction de l'importance de leur violation. Nous avons tenté de les trier en ordre décroissant de l'importance de leur violation, croyant que cette méthode aurait donné de meilleurs résultats. La technique utilisée s'est cependant révélée plus efficace, contrairement à l'intuition. En respectant l'ordre du tri, l'algorithme procède ensuite au rétablissement du respect des capacités un commutateur à la fois, en suivant la procédure itérative suivante. L'algorithme calcule premièrement pour chacune des cellules affectées au commutateur dont le respect des capacités est présentement rétabli, la différence entre les coûts (liaison et relèves) relatifs audit commutateur (qui constitue le commutateur optimal pour ces cellules, la capacité n'ayant pas été respectée lors de l'exécution de l'heuristique d'OCF) et ceux relatifs au commutateur disposant des ressources suffisantes ayant les deuxièmes coûts les moins élevés. Il ré-affecte ensuite la cellule dont le produit $1/DiffCoûtsTotaux*NbAppels$ est le plus élevé. Cette formule permet de tenir compte non seulement des coûts de liaison et de relèves, mais également du nombre d'appels afin de choisir la prochaine cellule à ré-affecter. Nous tentons ainsi de rétablir le respect des capacités des commutateurs avec le moins de ré-affectations possibles, tout en effectuant les mouvements les moins coûteux. La cellule sélectionnée est ré-affectée au commutateur ayant les deuxièmes coûts les moins élevés. Cette procédure se termine lorsque la capacité de tous les commutateurs du réseau a été rétablie.

4.2.3 Troisième variante k-opt

Cette troisième variante est similaire à la deuxième et s'applique également lorsque l'algorithme n'était pas tenu de respecter la capacité des commutateurs lors de l'exécution de l'heuristique d'OCF. Cette variante est cependant divisée en deux étapes.

Lors de l'exécution de la première étape, l'algorithme applique la même procédure que celle décrite ci-dessus en ce qui concerne la deuxième variante, mais a cependant la possibilité de violer la capacité des commutateurs disposant au départ de ressources suffisantes (les cellules ne peuvent toutefois pas être affectées à un commutateur dont la capacité n'est déjà pas respectée au démarrage de l'étape). Il devra toutefois rétablir cette capacité lors de l'exécution de la deuxième étape, qui est identique à la deuxième variante k-opt. Nous espérons avec cette technique que l'augmentation des coûts relatifs aux mouvements effectués lors de la première étape qui autorise les cellules à être affectées à des commutateurs tiers sans tenir compte de leur ressources (consacrée au rétablissement du respect des capacités des commutateurs dont la capacité a été violée lors de la phase d'OCF), ainsi que les mouvements effectués lors de la deuxième étape (consacrée au rétablissement de la capacité des commutateurs violée lors de la deuxième étape) sera inférieure à la deuxième variante k-opt. En ayant la possibilité de violer la capacité des commutateurs disposant des ressources suffisantes au début de la première étape, l'algorithme pourra ainsi parfois procéder à des affectations qu'il eût été impossibles de réaliser en respectant lesdites capacités. Ces mouvements généreront éventuellement eux-mêmes de nouvelles violations (concernant de nouveaux commutateurs, la capacité des premiers commutateurs ayant été rétablie). L'algorithme éliminera ces nouvelles violations au cours de la seconde phase. Pour que la technique soit jugée efficace, l'ensemble de ces mouvements devra procurer une solution ayant un coût inférieur à la deuxième variante k-opt.

4.3 Résultats de la méta-heuristique d'OCF

Nous constatons en ce qui concerne le problème d'affectation unique, que l'heuristique d'OCF est «victime» de l'efficacité de l'implémentation de l'algorithme et de la mise en œuvre des règles précédemment décrites d'une part et de la série des données utilisées pour les tests d'autre part. L'ensemble des règles utilisées au moment de la sélection de la prochaine cellule à affecter ainsi que les règles employées afin de déterminer à quel commutateur affecter ladite cellule étant relativement efficace en

elles-mêmes, elles ont pour conséquence de fournir une solution se rapprochant des meilleurs résultats obtenus avec l'ensemble des méthodes implémentées sans avoir recours à la mise en œuvre des paramètres de la méthode d'OCF. L'analyse des données utilisées pour effectuer les tests relatifs au problème d'affectation unique nous permet de constater par ailleurs que la proportion des coûts de relèves par rapport aux coûts de liaison est insignifiante. Les coûts de liaison totaux concernant les données du test numéro 10 pour 200 cellules et 7 commutateurs s'élèvent par exemple à la somme de 56.871,20 alors que les coûts de relèves ne représentent que 333,54 (soit 0,0058 % - ce pourcentage étant encore moins important pour certaines séries de données). Aussi, il y a de manière générale une différence relativement importante entre les coûts de liaison relatifs au commutateur le moins coûteux et ceux relatifs au commutateur le deuxième moins coûteux pour chacune des cellules (cette différence étant généralement beaucoup plus importante que l'ensemble des coûts de relèves pouvant intervenir entre les cellules contiguës).

Ainsi, l'algorithme se retrouve dans le cas du problème d'affectation unique en présence de données lui permettant d'effectuer son choix d'affectation en se basant presque uniquement sur les coûts de liaison (simplifiant considérablement sa tâche) et en se basant sur la différence des coûts entre les commutateurs les moins coûteux afin d'établir l'ordre de sélection des cellules. Cette situation ne se retrouve pas dans le cas des instances du problème de double affectation pour lesquelles la proportion des coûts de relèves est relativement importante par rapport aux coûts de liaison, comme nous le constaterons lors de l'analyse des résultats relatifs à la méthode de domiciliation double. Notons également que la différence des coûts entre les deux commutateurs les moins coûteux est beaucoup moins importante en ce qui concerne les instances de ce problème. La mise en œuvre de nos paramètres lors de l'exécution de la méthode consacrée à la résolution de ce problème aura ainsi une influence sur l'efficacité de l'algorithme ainsi que sur la qualité de la solution finale. Soulignons que les données utilisées pour effectuer les tests concernant ce problème de domiciliation double ont été obtenues au moyen du générateur de données tests utilisé par Merchant et Sengupta (les données

ayant été générées par André et al.). Ce générateur attribue aux coûts de relèves des valeurs similaires à celles attribuées aux coûts de liaison. Les fichiers utilisées pour effectuer les tests concernant le problème d'affectation unique ont de leur côté été créés par Houéto et al., et comprennent des coûts de relèves qui ont en général 1 centième de la valeur des coûts de liaison. Il est donc normal que l'algorithme se comporte différemment en présence des fichiers utilisés pour l'exécution des tests concernant l'affectation unique par rapport à lorsqu'il utilise les fichiers générés pour le problème de domiciliation double.

Ainsi, dans la mesure où les résultats obtenus en n'utilisant qu'une seule fourmi, en n'exécutant qu'un seul cycle (sans autoriser l'algorithme de tenter d'améliorer la solution du premier cycle) et une seule boucle sont comparables à ceux obtenus par l'ensemble des techniques développées pour la résolution de notre problème d'affectation unique, la mise en œuvre des paramètres (autorisant plusieurs fourmis, un nombre de cycles et de boucles supérieurs à 1) n'a pas pour effet d'augmenter de manière systématique et significative la qualité de la solution à notre problème.

Les Tableaux 4.4 et 4.5 présentent les résultats comparatifs en ce qui concerne l'exécution de la variante 1/1/0, pour deux instances comportant respectivement 30 cellules et 3 commutateurs, et 200 cellules et 7 commutateurs avec les paramètres fixés à 1/1/100/100/0/0/1/0/0, 7/2/90/100/0/0/3/5.5/5.5 et 7/2/90/100/0.1/0/3/5.5/5.5 successivement. Le Tableau 4.2 décrit la signification des indices servant à distinguer les différentes variantes.

Tableau 4.2 Description des indices servant à distinguer les différentes variantes

Indice	Fonction
1	Règle d'affectation utilisée : 1 = Moindre Coût ; 2 = Moindre Augmentation
2	Respect systématique ou violation temporaire autorisée de la capacité des commutateurs : 1 = Respect systématique ; 0 = violation temporaire autorisée
3	Méthode k-opt de recherche locale utilisée : 1, 2 et 3 correspondant aux trois méthodes décrites précédemment ; 0 signifiant que l'heuristique d'OCF est utilisée seule, sans recours à une technique de recherche locale

Ainsi, dans note exemple,

- le premier 1 signifie que l'alternative *Moindre Coût* a été utilisée,
- le second 1 signifie que la capacité des commutateurs a dû être systématiquement respectée,
- et le 0 signifie qu'aucune recherche locale k-opt n'a été employée.

Le Tableau 4.3 décrit la signification des indices servant à distinguer les différents paramètres utilisés.

Tableau 4.3 Description des indices servant à distinguer les différants paramètres

Indice	Fonction
1	Nombre de fourmis utilisé
2	Nombre de cycles sans amélioration autorisé
3	Valeur du paramètre *Parq0Sel*, relatif à la sélection
4	Valeur du paramètre *Parq0Att*, relatif à l'affectation
5	Valeur du paramètre *ParAlphaSel*, concernant la sélection de la prochaine cellule à affecter, le paramètre β étant calculé par différence
6	Valeur du paramètre *ParAlphaAtt*, concernant l'affectation, le paramètre β étant calculé par différence
7	Nombre de boucles devant être exécuté
8	Quantité de phéromones déposée par l'ensemble des fourmis
9	Quantité de phéromones déposée par la fourmi privilégiée

Ainsi, les paramètres se lisent de la manière suivante, en respectant l'ordre séquentiel, pour la première série (1/1/100/100/0/0/1/0/0) :

- 1 = une seule fourmi est utilisée ;
- 1 = un seul cycle est autorisé ;
- 100 = l'ordre de sélection est déterminé uniquement par l'application du premier argument de la règle pseudo-aléatoire proportionnelle (aucun élément probabiliste) ;
- 100 = l'affectation est déterminée uniquement par l'application du premier argument de la règle pseudo-aléatoire proportionnelle (aucun élément probabiliste) ;

- 0 = le poids attribué aux traînées de phéromones concernant l'ordre de sélection est nul (seuls les données du problème sont prises en compte- correspondant aux règles de sélection) ;
- 0 = le poids attribué aux traînées de phéromones concernant l'affectation est nul (seules les données du problème sont prises en compte- correspondant aux règles d'affectation) ;
- 1 = une seule boucle doit être exécutée ;
- 0 = aucune fourmi ne peut déposer des phéromones ;
- 0 = aucune fourmi privilégiée ne peut déposer des phéromones.

Les séries suivantes se lisent de manière similaire.

Tableau 4.4 Résultats comparatifs de la variante 1/1/0 pour les paramètres fixés à différentes valeurs (30 cellules et 3 commutateurs)

		ACO "1/1/0" 7/1/90/100/0/0/3/5.5/5.5			ACO "1/1/0" 7/2/90/100/0/0/3/5.5/5.5		
30-3	1	[illegible]	[illegible]	[illegible]	0.02	0.03	302
	10	[illegible]	[illegible]	[illegible]	0.02	0.03	339
	11	[illegible]	[illegible]	[illegible]	0.02	0.03	273
	12	[illegible]	[illegible]	[illegible]	0.02	0.03	300
	13	[illegible]	[illegible]	[illegible]	0.02	0.03	301
	14	[illegible]	[illegible]	[illegible]	0.02	0.03	383
	15	[illegible]	[illegible]	[illegible]	0.02	0.03	357
	16	[illegible]	[illegible]	[illegible]	0.02	0.03	273
	17	[illegible]	[illegible]	[illegible]	0.02	0.02	282
	18	[illegible]	[illegible]	[illegible]	0.02	0.02	316
	19	[illegible]	[illegible]	[illegible]	0.03	0.03	397
	20	[illegible]	[illegible]	[illegible]	0.02	0.03	269
	21	[illegible]	[illegible]	[illegible]	0.02	0.03	305
	22	[illegible]	[illegible]	[illegible]	0.02	0.03	254
	23	[illegible]	[illegible]	[illegible]	0.02	0.03	378
							4729

		ACO "1/1/0" 7/2/90/100/0.1/0/3/5.5/5.5		
30-3	1	0.06	0.06	302
	10	0.00	0.00	338
	11	0.00	0.05	273
	12	0.00	0.00	301
	13	0.00	0.00	301
	14	0.05	0.05	383
	15	0.00	0.00	347
	16	0.06	0.06	273
	17	0.00	0.00	283
	18	0.05	0.05	316
	19	0.00	0.00	397
	20	0.00	0.06	269
	21	0.00	0.00	305
	22	0.00	0.00	252
	23	0.06	0.06	372
				4712

Tableau 4.5 Résultats comparatifs de la variante 1/1/0 pour les paramètres fixés à différentes valeurs (200 cellules et 7 commutateurs)

			ACO "1/1/0" 7/2/90/100/0/0/3/5.5/5.5			ACO "1/1/0" 7/2/90/100/0.1/0/3/5.5/5.5		
200~7	1	[illegible]	0.83	1.09	4052	0.83	1.05	4011
	10	[illegible]	0.83	1.10	3006	0.88	1.10	3017
	11	[illegible]	0.84	1.10	2975	0.87	1.09	2963
	12	[illegible]	0.83	1.09	3311	0.83	1.05	3312
	13	[illegible]	0.83	1.10	3383	0.88	1.10	3375
	14	[illegible]	0.83	1.10	3329	0.88	1.10	3334
	15	[illegible]	0.83	1.10	3026	0.82	1.04	3027
	16	[illegible]	0.83	1.09	3368	0.88	1.10	3379
	17	[illegible]	0.83	1.09	3124	0.88	1.10	3086
	18	[illegible]	0.83	1.10	3124	0.82	1.04	3162
	19	[illegible]	0.84	1.10	4027	0.88	1.10	4038
	20	[illegible]	0.84	1.10	3692	0.88	1.10	3655
	21	[illegible]	0.83	1.09	2999	0.82	1.04	2992
	22	[illegible]	0.83	1.10	3138	0.88	1.10	3177
	23	[illegible]	0.83	1.10	3026	0.88	1.10	3029
		[illegible]			49582			49557

Nous constatons ainsi une diminution de -0.0034 ((4729-4745)/4745) et -0.0069 ((4712-4745)/ 4745) des coûts (les données correspondent à la sommation de l'ensemble des instances retrouvée au bas des tableaux) pour les paramètres fixés respectivement à 7/2/90/100/0/0/3/5.5/5.5 et 7/2/90/100/0.1/0/3/5.5/5.5, par rapport à la méthode 1/1/100/100/0/0/1/0/0 n'admettant qu'une fourmi, qu'un cycle et une boucle et ne comportant aucun élément probabiliste, en ce qui concerne l'ensemble des problèmes comprenant 30 cellules et 3 commutateurs. Ces diminutions sont de -0.0063 et -0.0068 en ce qui concerne les mêmes paramètres pour les problèmes de 200 cellules et 7 commutateurs (le temps d'exécution augmentant parallèlement de plus de 8 dixièmes de secondes).

Nous présentons uniquement les résultats obtenus en intégrant un aspect probabiliste en ce qui concerne l'ordre de sélection des cellules. La valeur du paramètre *Parq0Sel* correspondant à l'application soit du premier argument (qui consiste à choisir

la cellule dont le *moindre regret* est le plus élevé), soit du deuxième argument (qui consiste en une sélection aléatoire) de la règle pseudo aléatoire proportionnelle, est fixée à 90 % dans les deux exemples décrit ci-dessus comportant un élément aléatoire. Le paramètre *ParAlphaSel* (α, le β étant calculé par différence) correspondant au poids accordé aux phéromones est fixé de son côté à 0.1 dans le deuxième exemple comportant un élément aléatoire (l'algorithme attribue ainsi un poids de 10 % aux phéromones et de 90 % aux données du problème). En attribuant une valeur relativement faible aux paramètres modifiables, l'algorithme utilise ainsi la plupart du temps les données du problème afin d'effectuer ses choix, mais intègre un aspect probabiliste dans sa recherche afin de la diversifier. Cette démarche nous donne de bons résultats comme nous l'avons mentionné ci-dessus (voir diminutions constatées préalablement).

Nous remarquons une diminution plus importante lorsque les deux paramètres (*Parq0Sel* et *ParAlphaSel*) sont modifiés. L'attribution de valeurs plus élevées à nos deux variables a pour effet d'augmenter le temps d'exécution inutilement et dégénère souvent la solution trouvée. L'augmentation du temps d'exécution s'explique par la moindre utilisation des règles de sélection utilisées en raison du caractère plus aléatoire de la recherche. Dans la mesure où l'algorithme effectue déjà ses recherches dans les zones avantageuses au moyen des règles de sélection, il effectue souvent inutilement des recherches dans des zones plus coûteuses en le dirigeant vers ces autres zones au hasard. Il doit par ailleurs mettre un certain temps avant de converger à nouveau vers les zones contenant les meilleures solutions lorsqu'il a initialement été dirigé vers ces zones moins favorables. Cette convergence prend enfin parfois du temps à se réaliser puisque le paramètre aléatoire continue d'imposer son influence tout au long de l'exécution de l'algorithme. En attribuant au contraire une faible valeur aux éléments aléatoire de l'heuristique, l'algorithme suit de manière globale son cours «déterministe» habituel (dicté par les données du problème), mais diversifie un peu plus sa recherche, lui permettant de rencontrer de meilleures solutions.

L'intégration d'un élément aléatoire en ce qui concerne les paramètres d'affectation donne de moins bons résultats puisque la règle «déterministe» consistant à

affecter la cellule au commutateur de *moindre coût* (ou de *moindre augmentation*) est relativement efficace en elle-même. À chaque fois que l'algorithme attribuerait une cellule à un commutateur non optimal en raison de l'application d'une règle aléatoire, il aurait pour effet d'augmenter les coûts de configuration du réseau. Rappelons que les cellules sont attribuées à leur commutateur en ordre des *moindres regrets*. Ainsi, lorsque l'algorithme approche la fin des affectations à effectuer et qu'il est obligé d'attribuer une cellule à un commutateur non optimal (en raison de l'épuisement de ses capacités) il contient dans son ensemble de cellules non encore affectées, les cellules ayant une différence moins élevée entre les deux commutateurs de moindres coûts (en raison de l'application de la règle des *moindres regrets*). Rappelons également que lorsqu'une cellule ne peut être attribuée à son commutateur optimal, elle est affectée à son commutateur pour lequel les coûts sont les deuxièmes (ou troisième...) moins élevés.

4.4 Hybridation de la méthode *k*-opt avec l'heuristique d'OCF

L'utilisation de l'heuristique d'OCF employée seule donne des résultats similaires (et parfois très légèrement meilleurs) à ceux obtenus avec l'utilisation combinée de la technique de recherche locale k-opt en ce qui concerne les problèmes de petite taille (30 cellules et 3 commutateurs). Cette situation peut s'expliquer en raison de la simplicité de ces problèmes qui se résolvent efficacement uniquement à l'aide des règles de sélection et d'affectation utilisées par l'heuristique d'OCF. En ce qui concerne les problèmes plus complexes de taille importante (200 cellules et 7 commutateurs) les résultats obtenus avec la combinaison de la technique k-opt sont meilleurs comme l'illustre les tableaux 4.6 et 4.7.

Tableau 4.6 Comparaison des variantes 1/1/0 & 2/1/0 et 1/0/2, avec l'ensemble des paramètres fixés à une valeur «nulle» (200 cellules et 7 commutateurs)

		ACO "1/1/0"			ACO "2/1/0"			[illegible]		
200-7	1	0.02	0.28	4041	0.03	0.30	4041	[illegible]		
	10	0.01	0.27	3048	0.04	0.30	3035	[illegible]		
	11	0.01	0.28	2975	0.04	0.30	2975	[illegible]		
	12	0.01	0.27	3353	0.03	0.29	3353	[illegible]		
	13	0.01	0.27	3375	0.03	0.29	3375	[illegible]		
	14	0.01	0.28	3327	0.03	0.30	3328	[illegible]		
	15	0.01	0.27	3026	0.03	0.30	3024	[illegible]		
	16	0.01	0.27	3356	0.03	0.30	3342	[illegible]		
	17	0.01	0.28	3138	0.03	0.29	3121	[illegible]		
	18	0.01	0.27	3214	0.03	0.29	3214	[illegible]		
	19	0.01	0.28	4080	0.03	0.30	4080	[illegible]		
	20	0.02	0.28	3681	0.03	0.29	3665	[illegible]		
	21	0.01	0.27	2998	0.03	0.29	2997	[illegible]		
	22	0.01	0.28	3257	0.03	0.30	3313	[illegible]		
	23	0.01	0.27	3029	0.04	0.30	3027	[illegible]		
				49698			49690			[illegible]

Tableau 4.7 Comparaison des variantes 1/1/0 & 2/1/0 et 1/0/3, avec l'ensemble des paramètres fixés à une valeur «nulle» (200 cellules et 7 commutateurs)

		ACO "1/1/0"			ACO "2/1/0"			ACO/Kopt "1/0/3"		
200-7	1	0.02	0.28	4041	0.03	0.30	4041	0.01	0.28	3960
	10	0.01	0.27	3048	0.04	0.30	3035	0.02	0.29	2995
	11	0.01	0.28	2975	0.04	0.30	2975	0.01	0.27	2969
	12	0.01	0.27	3353	0.03	0.29	3353	0.01	0.28	3287
	13	0.01	0.27	3375	0.03	0.29	3375	0.01	0.28	3371
	14	0.01	0.28	3327	0.03	0.30	3328	0.01	0.27	3335
	15	0.01	0.27	3026	0.03	0.30	3024	0.01	0.27	3035
	16	0.01	0.27	3356	0.03	0.30	3342	0.02	0.28	3291
	17	0.01	0.28	3138	0.03	0.29	3121	0.01	0.27	3104
	18	0.01	0.27	3214	0.03	0.29	3214	0.01	0.27	3134
	19	0.01	0.28	4080	0.03	0.30	4080	0.01	0.28	3886
	20	0.02	0.28	3681	0.03	0.29	3665	0.01	0.27	3589
	21	0.01	0.27	2998	0.03	0.29	2997	0.01	0.27	2997
	22	0.01	0.28	3257	0.03	0.30	3313	0.01	0.28	3079
	23	0.01	0.27	3029	0.04	0.30	3027	0.02	0.28	3035
				49698			49690			49067

Soulignons que les temps d'exécution en utilisant l'heuristique d'OCF seule et en ayant recours à la combinaison des deux techniques sont presque identiques. La variante consistant à utiliser la *moindre augmentation* en ce qui concerne l'affectation (au cours de l'heuristique d'OCF) est légèrement plus efficace que la technique consistant à utiliser le *moindre coût* en ce qui concerne les instances de grande taille, lorsque l'heuristique d'OCF est utilisée seule ou en combinaison avec la première variante de la technique k-opt (soulignons que cette alternative - moindre augmentation - ne peut être utilisée que lorsque la capacité des commutateurs doit être systématiquement respectée.).

L'algorithme donnant les meilleurs résultats en ce qui concerne les problèmes de grande taille lorsque l'ensemble des paramètres de l'heuristique d'OCF sont fixés à une valeur «nulle», est celui appliquant la variante 1/0/2 (1 = Moindres Coûts/0 = Non-Respect des Capacités/2 = Variante k-opt 2). Les résultats de cette technique ont été reportés aux Tableaux 4.8 et 4.9 (en ce qui concerne les instances de 200 cellules et 7 commutateurs), consacrés à la comparaison de notre méthode avec certaines développées antérieurement. Le graphique correspondant aux données du tableau comparant la technique ayant donné les meilleurs résultats (RT/PC) avec notre meilleure variante (1/0/2) est représenté à la Figure 4.3. Chaque instance est reportée individuellement. Les temps d'exécution de ces deux méthodes sont comparés à la Figure 4.4. L'Annexe B présente les résultats comparatifs de ces mêmes méthodes pour des instances de 30 cellules et 3 commutateurs ainsi que pour 100 cellules et 5 commutateurs.

Tableau 4.8 Résultats comparatifs entre les méthodes RT-PC et PC employée seule (André et al.), et la méthode d'OCF-2ème variante k-opt

		RT/VNS (Sol. Init PC)		PC (Sol. Init)					Diff (%) RT/VNS	Diff (%) PC
200~7	1	3930	4.59	3961	0.23				0.0081	-0.0048
	10	2985	2.36	3006	0.18				0.0117	0.0040
	11	2956	1.48	2972	0.21				0.0014	-0.0040
	12	3285	2.81	3296	0.23				0.0079	0.0046
	13	3361	2.66	3365	0.21				0.0030	0.0018
	14	3326	1.34	3327	0.23				0.0027	0.0024
	15	3021	1.18	3034	0.22				0.0046	0.0003
	16	3285	3.69	3287	0.22				0.0018	0.0012
	17	3077	3.16	3080	0.26				0.0023	0.0013
	18	3054	4.19	3143	0.23				0.0262	-0.0029
	19	3617	5.53	3693	0.26				0.0213	-0.0511
	20	3432	6.51	3550	0.23				0.0178	-0.0161
	21	2982	2.38	2987	0.23				0.0148	0.0131
	22	3053	4.19	3079	0.24				0.0396	0.0309
	23	3027	0.77	3039	0.29				0.0026	-0.0013
		48391		49041					0.0110	-0.0024

Tableau 4.9 Résultats comparatifs entre les méthodes RT-Chaînes d'éjection (André et al.) et RT (Houéto et al.), et la méthode d'OCF-2ème variante k-opt

		RT (Houéto)		RT/Chaine Eject (CE)					Diff (%) (Houéto)	Diff (%) (CE)
200~7	1	4081	2.58	3935	0.34				-0.0292	0.0069
	10	3021	2.16	2986	0.36				-0.0003	0.0114
	11	2969	3.16	2956	0.29				-0.0030	0.0014
	12	3357	2.30	3283	0.35				-0.0137	0.0085
	13	3397	2.54	3360	0.34				-0.0077	0.0033
	14	3343	2.69	3326	0.30				-0.0024	0.0027
	15	3027	3.54	3021	0.30				0.0026	0.0046
	16	3300	3.09	3283	0.29				-0.0027	0.0024
	17	3120	2.76	3075	0.31				-0.0115	0.0029
	18	3163	2.63	3068	0.33				-0.0092	0.0215
	19	3813	4.99	3641	0.41				-0.0312	0.0146
	20	3531	2.81	3425	0.36				-0.0106	0.0199
	21	3053	2.11	2982	0.31				-0.0068	0.0148
	22	3126	2.57	3053	0.32				0.0154	0.0396
	23	3028	2.50	3027	0.33				0.0023	0.0026
		49329		48421					-0.0062	0.0104

Les tableaux comportent la différence en pourcentage des résultats des méthodes comparées et de ceux de notre variante 1/0/2. Les premiers temps d'exécution de nos méthodes (exprimés en secondes) concernent uniquement l'exécution de l'algorithme lui-même, sans tenir compte de la lecture des données sur disque. Les deuxièmes temps intègrent le temps nécessaire à cette lecture. Le temps global mis par l'algorithme afin d'effectuer l'ensemble des 75 tests (15*30 cellules et 3 commutateurs, 15*50 cellules et 4 commutateurs, 15*100 cellules et 5 commutateurs, 15*150 cellules 6 commutateurs, 15*200 cellules et 7 commutateurs) est de 8,192 secondes pour la technique 1/0/2 (ce temps comprend le temps de lecture).

Parmi les méthodes servant de comparaison, la technique ayant donné les meilleurs résultats est celle de recherche taboue avec solution initiale fournie par la technique de PC développée par André et al. La somme de l'ensemble des 15 tests concernant le problème de 200 cellules et 7 commutateurs s'élève à 48.391. La somme des résultats de notre meilleure technique (1/0/2) concernant ces mêmes tests est de 48.925, soit une différence de 1,10 %. Toutefois, le temps d'exécution nécessaire à l'obtention de l'ensemble des résultats (pour 200 cellules et 7 commutateurs) de l'heuristique RT/PC est de 46,84 secondes alors qu'il n'est que de 0,17 secondes sans tenir compte de la lecture et de 4,13 secondes en tenant compte de cette lecture pour l'heuristique OCF/k-opt. En ce qui concerne la technique de PC employée seule, cette méthode fournit des résultats s'élevant à la somme de 49.041 en 3,47 secondes. Les résultats de notre méthode sont inférieurs de 0,24 % (soit 48.925) par rapport à cette technique. La recherche taboue employée avec une solution générée autrement qu'avec la technique de PC fournit en 95,44 secondes une solution inférieure à la nôtre de 1,07 %, alors que la même méthode utilisée avec la technique des chaînes d'éjection génère une solution inférieure à la nôtre de 1,04 % en 4,94 secondes. Enfin, l'heuristique de recherche taboue développée par Houéto et Pierre procure une solution qui est supérieure à la nôtre de 0,82 % en 42,43 secondes.

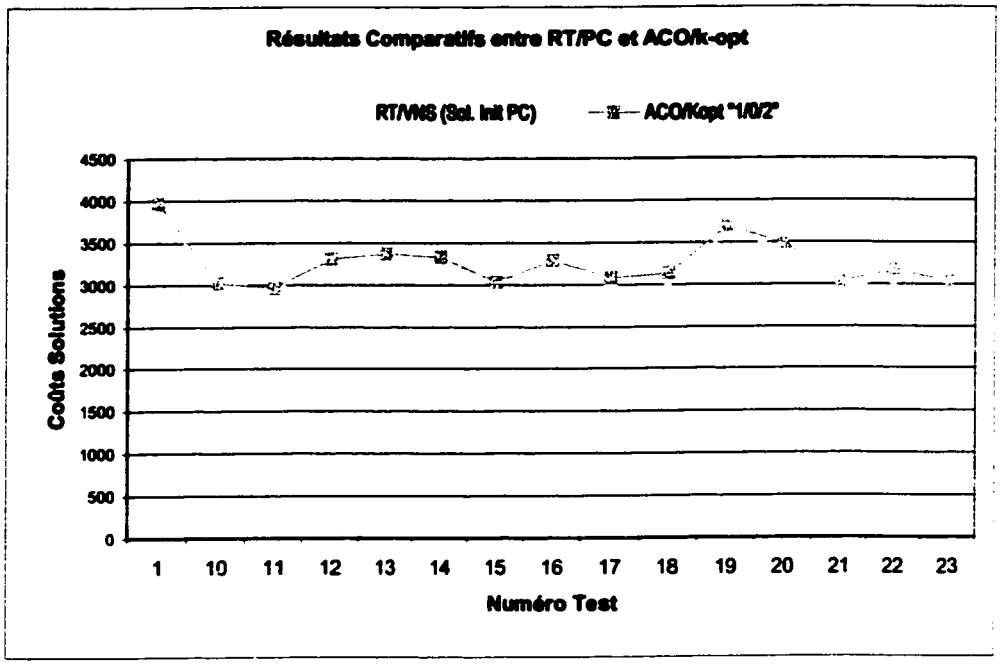

Figure 4.3 Résultats comparatifs entre les méthodes RT/PC et OCF/k-opt

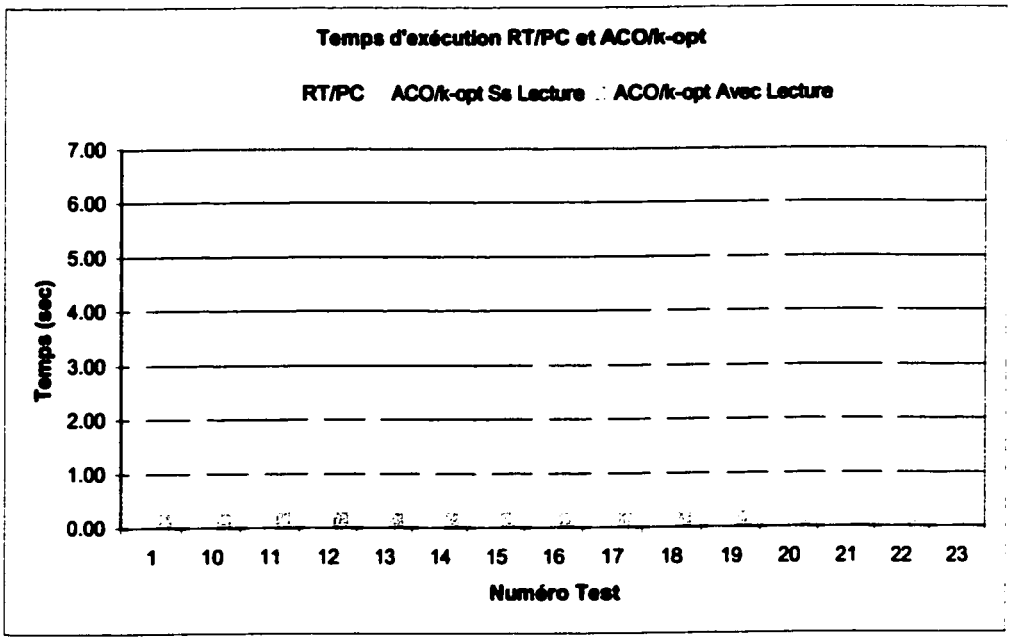

Figure 4.4 Temps d'exécution des méthodes RT/PC et OCF/k-opt

Nous obtenons des résultats similaires en ce qui concerne les instances de problèmes de petites (30 cellules et 3 commutateurs) et moyennes tailles (100 cellules et 5 commutateurs) comme indiqué dans l'Annexe B. Nous constatons cependant que la technique est de manière générale très légèrement moins efficace lorsque la taille des instances est réduite. L'efficacité de l'algorithme augmente proportionnellement avec la taille des instances, comparativement aux autres méthodes implémentées. Par ailleurs, l'alternative 1/1/0 procure en général et de manière très marginale, de meilleurs résultats à ceux de l'alternative 1/0/2 pour les instances de petites tailles.

Soulignons que le temps d'exécution de notre variante 1/0/2 se situe systématiquement entre 1 et 2 centièmes de seconde pour chacune des instances du problème comprenant 200 cellules et 7 commutateurs si nous attribuons à l'ensemble des paramètres de l'heuristique une valeur «nulle», et si nous ne tenons pas compte de la lecture des données sur disques. La complexité de l'algorithme lorsque l'ensemble de ces paramètres sont fixés à une valeur «nulle» est de $O(n*n)$, l'algorithme n'appelant qu'une fourmi, n'exécutant qu'un seul cycle et une seule boucle. Ainsi, le temps d'exécution de notre algorithme est très stable et n'augmente que par un facteur $n*n$ lorsque la taille des instances devient plus importante. Il serait donc possible d'utiliser l'algorithme sur des instances comprenant un nombre de cellules et de commutateurs beaucoup plus important sans accroître le temps d'exécution par un facteur supérieur à $n*n$. La qualité des solutions générées serait par ailleurs théoriquement similaire.

Nous avons enfin effectué quelques exécutions prolongées de l'algorithme afin d'analyser les résultats. L'Annexe A présente les résultats comparatifs entre la méthode RT-PC d'André et al., ceux de notre variante OCF 1/0/2 exécutée avec l'ensemble des paramètres fixé à une valeur «nulle» ainsi que les résultats de l'exécution de cette même variante avec ses paramètres fixés aux valeurs suivantes : 7/7/95/100/0.1/0/0/5.5/5.5 (7 fourmis/7 cycles sans amélioration/*Parq0Sel* fixé à 95/*Parq0Att* fixé à 100/*ParAlphaSel* fixé à 0.1/*ParAlphaAtt* fixé à 0/*NbBoucles* = *Nb_Cellules*/*Nb_Fourmis*). Le temps nécessaire à l'algorithme afin d'exécuter cette alternative avec ces paramètres s'est élevé à 4,45 secondes pour les 15 instances comprenant 200 cellules et 7 commutateurs. Les

résultats obtenus sont légèrement meilleurs que ceux générés lorsque l'algorithme est exécuté en utilisant des paramètres fixés à une valeur «nulle», tel que le tout est constaté à l'Annexe A. Nous avons également permis à l'algorithme d'exécuter un nombre de cycles supérieurs à 7. Nous constatons toutefois que les résultats obtenus sont similaires et n'améliorent pas les solutions obtenues.

4.5 Applications au problème de domiciliation double

Nous avons effectué une implémentation simplifiée de notre heuristique au problème de domiciliation double afin de ne pas augmenter la complexité du problème. Notre algorithme applique itérativement ses techniques développées pour le problème d'affectation unique aux deux séries de données correspondant aux deux périodes de la journée telles que définies précédemment, en ayant soin cependant de ne pas comptabiliser les coûts de liaison à deux reprises lorsqu'une cellule est affectée au même commutateur au cours des deux périodes. Notons qu'il serait possible d'augmenter le nombre d'itérations effectué par l'algorithme si l'on désirerait tenir compte de plus de deux périodes. Cette manière de procéder est tout à fait réaliste.

Supposons par exemple que l'on dispose de deux (ou davantage) séries de données des différentes périodes, correspondant aux communications constatées ou prévues dans un réseau devant être conçu ou modifié. On demandera à l'algorithme de réitérer son exécution autant de fois qu'il y a de périodes devant être prises en compte. Chaque période peut être considérée comme un problème indépendant, en s'assurant bien entendu de ne pas comptabiliser les coûts de liaison d'une cellule au même commutateur à plusieurs reprises. Lorsqu'une cellule a été affectée à un commutateur lors d'une période précédente, il suffit d'attribuer une valeur nulle correspondant au coût de liaison entre cette cellule et le commutateur en question avant de procéder aux affectations relatives aux périodes subséquentes. Il est par ailleurs possible d'inverser l'ordre de traitement des différentes périodes afin de trouver la meilleure solution. Dans la mesure où les affectations effectuées au cours des itérations ont une influence sur les affectations des boucles subséquentes (en attribuant une valeur nulle à certains coûts de

liaison lors de l'initialisation des itérations successives) les solutions générées par l'algorithme seront sans doute différentes en modifiant l'ordre de traitement des différentes périodes. Après avoir généré l'ensemble des affectations pour chacune des périodes considérées, il sera possible d'implémenter un algorithme dynamique servant à déterminer en temps réel les moments opportuns pour effectuer les changements d'affectation concernant la gestion des cellules disposant de plusieurs liaisons physiques. Notre implémentation ne tient compte que de l'affectation statique concernant les liens physiques à établir entre chacune des cellules et l'ensemble des commutateurs du réseau.

Nous sommes heureux de constater que les valeurs des paramètres de notre heuristique ont, en ce qui concerne ce problème de domiciliation double, une influence notable sur les résultats générés. Rappelons que les données utilisées pour ce problème sont différentes de celles employées pour effectuer les tests relatifs au problème d'affectation unique. Les coûts relatifs aux relèves sont remarquablement plus importants proportionnellement aux coûts de liaison, comparativement aux données utilisées pour le problème d'affectation simple. Aussi, les coûts de liaison n'ont pas une différence notable entre les différents commutateurs pour chaque cellule, contrairement aux instances du problème de domiciliation simple. Les résultats de ce problème de domiciliation double sont reproduits aux Tableaux 4.10 à 4.12. Les Tableaux 4.10 et 4.11 comparent l'ensemble des variantes de notre heuristique entre elles-mêmes en fixant l'ensemble des paramètres à une valeur «nulle» (1 seule fourmi, 1 seul cycle, 1 seule boucle, aucun élément aléatoire pour la sélection ou l'affectation, un poids d'une unité concernant les paramètres alpha et bêta et aucune phéromone déposée), alors que le Tableau 4.12 illustre les résultats de notre algorithme en accordant une valeur positive à certains paramètres modifiables.

Tableau 4.10 Comparaison des variantes 1/1/0 & 2/1/0 ainsi que 1/1/1 & 2/1/1 (100 cellules et 5 commutateurs) concernant le problème de domiciliation double

		ACO "2/1/0"				ACO/Kopt "1/1/1"				ACO/Kopt "2/1/1"			
100-5	1	0.03	0.18	208	160	0.01	0.16	206	194	0.03	0.18	208	208
	2	0.02	0.18	182	71	0.01	0.16	190	119	0.03	0.18	190	119
	3	0.03	0.18	161	160	0.01	0.16	161	208	0.03	0.18	161	208
	4	0.02	0.18	198	107	0.01	0.16	210	142	0.03	0.18	210	142
	5	0.03	0.18	198	128	0.01	0.16	215	195	0.03	0.17	215	184
	6	0.03	0.18	171	102	0.01	0.16	165	154	0.03	0.18	164	167
	7	0.03	0.18	208	96	0.01	0.16	229	147	0.03	0.18	208	150
	8	0.03	0.18	141	139	0.01	0.16	186	174	0.03	0.18	149	159
	9	0.02	0.17	139	80	0.01	0.16	136	144	0.03	0.18	136	144
	10	0.03	0.18	170	101	0.01	0.16	170	156	0.02	0.18	170	159
				2988	1844	1124			3499	1868	1631		3504	1869	1615

Tableau 4.11 Comparaison des variantes 1/0/2 et 1/0/3 (100 cellules et 5 commutateurs) concernant le problème de domiciliation double

		ACO/Kopt "1/0/3"			
100-5	1	0.01	0.16	333	292
	2	0.01	0.16	302	131
	3	0.01	0.16	273	321
	4	0.01	0.16	235	203
	5	0.01	0.16	419	254
	6	0.01	0.16	258	202
	7	0.01	0.16	343	145
	8	0.01	0.16	273	179
	9	0.01	0.17	174	180
	10	0.01	0.16	192	195
			4904	2802	2102

L'analyse des résultats reproduits aux Tableaux 4.10 et 4.11 démontre que l'utilisation de l'heuristique d'OCF employée seule donne de meilleurs résultats que lorsque l'algorithme combine la technique de recherche locale k-opt à sa recherche (notamment lorsqu'un droit de violation a temporairement été accordé). L'efficacité des règles utilisées par l'heuristique d'OCF est donc confirmée par ces résultats. Lorsque l'algorithme recourt à une des variantes k-opt afin de poursuivre sa recherche (ou rendre

admissible la solution trouvée par l'OCF dans le cas d'une variante admettant temporairement la violation), il met en œuvre de nouvelles règles qui, de toute évidence, sont moins efficaces et qui ont pour effet de diminuer la qualité de la solution obtenue par la technique d'OCF seule. Enfin, l'analyse de ces mêmes résultats, conjointement avec l'analyse d'autres résultats qui ne sont pas reproduits ici, indique que l'application de la technique de la *moindre augmentation* procure généralement de meilleurs résultats que celle consistant à appliquer le *moindre coût* lors de l'affectation, bien qu'il existe quelques exceptions (la variante 1/1/1 fournit de manière très marginale une meilleure solution par rapport à la variante 2/1/1 en ce qui concerne les résultats reproduits).

La variante 2/1/0 (2 = *Moindre Augmentation*/1 = *Respect des Capacités*/0 = *Aucune variante k-opt*) utilisant uniquement l'heuristique d'OCF sans recours à une recherche locale k-opt étant celle nous ayant procuré les meilleurs résultats en attribuant une valeur «nulle» à l'ensemble des paramètres modifiables de l'algorithme, nous l'avons utilisée pour exécuter notre algorithme en modifiant la valeur de certains paramètres. Les résultats de ces exécutions sont illustrés au Tableau 4.12 en comparaison avec l'exécution de l'algorithme lorsque tous les paramètres sont fixés à une valeur «nulle». Nous constatons à l'analyse des résultats reproduits, qu'en attribuant une faible valeur aux éléments probabilistes de l'algorithme (uniquement les paramètres relatifs à l'ordre de sélection des cellules, les paramètres relatifs à l'affectation devant rester «nul» lors de l'exécution de cette alternative de *moindre augmentation*), les coûts de configuration du réseau diminuent de manière considérable, autant pour les problèmes de petite taille que pour ceux de taille importante. Ces résultats confirment l'efficacité d'une recherche partiellement aléatoire concernant l'ordre de sélection, permettant à l'algorithme de diversifier sa recherche vers de nouvelles zones qui n'avaient pas été explorées en utilisant une démarche purement «déterministe», basée uniquement sur les données connus du problème. Lorsque la valeur des paramètres aléatoires est trop élevée, l'algorithme ne fournit pas d'aussi bons résultats, ne tenant plus suffisamment compte des données du problème.

Parmi les exécutions dont les résultats sont reproduits au Tableau 4.12, celle fournissant la meilleure solution est celle obtenue au moyen des paramètres 4/3/90/100/0.1/0/2/5.5/5.5 (procurant une diminution de 13,8 % par rapport à une exécution de l'algorithme avec tous les paramètres fixés à une valeur «nulle»), suivie de très près par la solution générée en appliquant les paramètres 3/2/90/100/0.1/0/2/5.5 5.5. Notons que les solutions reproduites au Tableau 4.12 ne sont pas nécessairement les meilleures pouvant être obtenues avec l'heuristique, mais ne servent qu'à démontrer l'efficacité de l'heuristique. Il est peut être possible d'obtenir de meilleurs résultats en modifiant la valeur des paramètres.

Tableau 4.12 Résultats comparatifs pour la variante 2/1/0 pour les paramètres fixés à différentes valeurs

		ACO "2/1/0"							ACO 2/1/0 3/2/95/100/0.1/0/2/5.5/5.5				ACO 2/1/0 3/2/90/100/0.1/0/2/5.5 5.5			
30~3	1	0.01	0.03	55	108				0.06	0.08	54	102	0.05	0.07	55	108
	2	0.00	0.02	73	56				0.09	0.11	56	32	0.06	0.08	65	39
	3	0.00	0.02	67	41				0.05	0.07	57	39	0.06	0.08	55	36
	4	0.00	0.02	27	176				0.05	0.07	27	131	0.06	0.08	27	131
	5	0.01	0.02	51	4				0.05	0.07	50	4	0.06	0.08	50	5
	6	0.00	0.02	55	165				0.06	0.08	43	125	0.06	0.08	39	123
	7	0.00	0.02	79	16				0.08	0.10	58	14	0.05	0.08	61	16
	8	0.00	0.02	63	85				0.06	0.08	44	53	0.07	0.09	44	52
	9	0.00	0.02	77	56				0.07	0.09	56	49	0.07	0.09	56	39
	10	0.01	0.02	48	104				0.06	0.10	48	89	0.07	0.09	46	68
				1405	595	810				1131	493	638		1115	498	617
100~5	1	0.03	0.18	206	150				0.79	0.95	188	125	0.93	1.08	191	154
	2	0.02	0.18	182	71				1.33	1.48	174	72	0.89	1.04	175	71
	3	0.03	0.18	161	150				0.98	1.13	169	122	0.96	1.11	157	117
	4	0.02	0.18	198	107				1.24	1.39	159	95	0.96	1.11	145	103
	5	0.03	0.18	188	128				1.29	1.44	173	107	0.94	1.09	151	115
	6	0.03	0.18	171	102				0.69	0.84	153	92	0.75	0.90	155	107
	7	0.03	0.18	286	96				1.06	1.22	191	85	0.96	1.11	185	83
	8	0.03	0.18	141	139				1.19	1.34	139	96	0.94	1.09	129	106
	9	0.02	0.17	139	80				0.83	0.98	127	82	1.04	1.19	127	76
	10	0.03	0.18	170	101				0.77	0.92	154	76	0.91	1.06	155	66
				2968	1844	1124				2579	1627	952		2568	1570	998

CHAPITRE V
CONCLUSION

La présente étude porte sur l'application de l'heuristique d'Optimisation par Colonie de Fourmis (OCF : ACO – Ant Colony Optimization) ainsi que de la méthode de recherche locale k-opt aux problèmes d'affectation unique et de domiciliation double des cellules aux commutateurs dans un réseau mobile. Nous effectuons une synthèse de nos travaux dans la première section de ce dernier chapitre. La deuxième section énonce les faiblesses relatives à nos méthodes ainsi qu'à leur implémentation. Enfin, nous terminons ce chapitre en énonçant dans une troisième section les travaux futurs qu'il serait possible d'entreprendre afin de poursuivre notre recherche à l'aide de nos méthodes, dans le but de les appliquer éventuellement à des problèmes connexes.

5.1 Synthèse de notre étude

Le problème d'affectation (unique ou double) de cellules aux commutateurs d'un réseau mobile est un problème NP-Complet. À titre de comparaison et afin d'illustrer la complexité du problème, les physiciens estiment le nombre de particules élémentaires dans notre univers observable à 10^{80}. L'instance de notre problème la plus complexe traitée contient 200 cellules et 7 commutateurs, admettant 7^{200} combinaisons d'affectations possibles (soit 1.046 E169 > 1.0 E80). Nous pouvons rappeler, afin de fournir une seconde comparaison, que le nombre de secondes s'étant écoulé depuis le commencement de notre Monde (si nous estimons l'âge de notre Univers à 15.5 milliards d'années) ne s'élève qu'à 4.25 E17 secondes. Il nous est nécessaire de recourir à une heuristique si nous souhaitons résoudre notre problème en un temps acceptable. Nous avons appliqué dans la présente étude la méta-heuristique relativement récente d'OCF audit problème, afin d'analyser son efficacité comparativement à d'autres méthodes qui avaient antérieurement été utilisées pour la résolution du même problème. Nous avons également combiné l'exécution de cette heuristique avec la technique d'optimisation locale k-opt afin de déterminer si elle apporterait une amélioration à la

première méthode. Plusieurs variantes des méthodes ont été créées dans le but de trouver la meilleure alternative possible. Les algorithmes développés ont été appliqués aux deux problèmes (affectation unique et domiciliation double) initialement décrits par les travaux de Merchant et Sengupta.

En comparaison avec les travaux précédemment effectués concernant le problème d'affectation unique, nous pouvons conclure que nos méthodes, ainsi que leur implémentation sont relativement efficaces, tant en ce qui concerne la qualité de la solution fournie qu'en ce qui concerne le temps d'exécution de l'algorithme. Les solutions fournies par notre meilleure variante (1/0/2) en ce qui concerne les problèmes de grande taille se situent en moyenne à un peu plus de 1 % pour un temps d'exécution inférieur (parfois important) par rapport aux autres méthodes ayant donné les meilleurs résultats et servant de comparaison (sauf pour la RT développée par Houéto et Pierre où nous obtenons de meilleurs résultats), et sont légèrement meilleures à la seule technique (PC employée seule) exécutée en un temps inférieur à la nôtre (si l'on prend en considération le temps de lecture, le temps d'exécution de ladite technique étant plus important que celui de notre méthode si l'on ne tient pas compte de la lecture de notre algorithme). Rappelons que la complexité de notre variante 1/0/2 est de l'ordre de $O(n^*n)$ lorsque les paramètres de l'heuristique d'OCF sont fixés à une valeur «nulle». Il serait ainsi possible d'augmenter la taille des instances du problème sans accroître le temps d'exécution de l'algorithme par un facteur supérieur à n^*n, tout en obtenant des solutions ayant une «qualité similaire».

En ce qui concerne le problème de domiciliation double, nous avons effectué une implémentation simplifiée du problème, consistant à appliquer l'algorithme développé pour l'affectation unique à chacune des deux périodes de la journée en s'assurant de ne pas comptabiliser les coûts de liaison à deux reprises lorsqu'une cellule est affectée au même commutateur pour les deux périodes. Rappelons qu'il serait possible d'appliquer l'algorithme à autant de périodes que l'on juge nécessaire. Nous constatons que les données utilisées afin d'effectuer les tests concernant ce problème de domiciliation double, comprenant une structure des coûts différente par rapport au problème

d'affectation unique (les coûts de relèves étant similaires aux coûts de liaison), nous incitent davantage à recourir à l'ajustement des paramètres de l'heuristique d'OCF comparativement au problème d'affectation unique, afin d'accroître la qualité des solutions obtenues. Notre algorithme nous permet cependant d'obtenir de bonnes solutions autant avec les fichiers créés pour effectuer les tests relatifs au problème d'affectation unique, qu'avec les fichiers générés pour le problème de domiciliation double. Ainsi, même si l'ajustement des paramètres de l'heuristique d'OCF n'a pas une influence considérable sur la qualité des résultats produits en ce qui concerne le problème d'affectation unique (avec les fichiers tests utilisés), l'algorithme fournit des solutions comparables à celles des meilleures solutions obtenues par l'ensemble des méthodes appliquées audit problème, uniquement à l'aide des règles de sélection d'ordre et d'affectation implémentées.

L'ajustement des paramètres de l'heuristique a cependant une influence favorable et remarquable lorsque la structure des coûts est semblable à celle retrouvée dans les fichiers produits pour le problème de domiciliation double, comme constaté par les résultats obtenus. Notons que nous avons effectué une série de tests relative au problème d'affectation unique, sur des données comportant une structure de coûts similaire à celle retrouvée dans les fichiers utilisés pour la double affectation afin d'analyser le comportement de l'heuristique. Nous constatons que, tout comme dans le cas du problème de domiciliation double, les paramètres de l'algorithme ont dans ce cas une influence importante sur la qualité des solutions. Les résultats de ces tests n'ont pas été reproduits dans la présente étude puisqu'il est possible d'effectuer la même analyse sur les résultats générés pour le problème de domiciliation double (l'algorithme relatif à ce problème étant en réalité identique à celui appliqué au problème d'affectation unique, sauf en ce qui concerne l'élimination des coûts de liaison lorsqu'une cellule est affectée au même commutateur au cours des deux périodes). Nous pouvons donc en conclure qu'il est possible d'utiliser favorablement notre algorithme avec des fichiers comprenant divers types de structure de coûts. Lorsque cette structure se rapprochera plutôt de celle retrouvée dans les fichiers générés pour le problème d'affectation unique, l'algorithme

utilisera notamment les règles de sélection afin de fournir les meilleures solutions, alors qu'il mettra davantage en œuvre les paramètres de l'heuristique d'OCF afin d'obtenir de meilleurs résultats lorsque la structure des coûts sera similaire à celle retrouvée dans les fichiers utilisés pour la domiciliation double.

5.2 Faiblesses de nos méthodes et de leur implémentation

Notre algorithme fournit de bons résultats (tant en ce qui concerne le temps d'exécution qu'en ce qui concerne la qualité des solutions obtenues) en attribuant certaines valeurs à certains paramètres de l'heuristique d'OCF. Toutefois, la méthode manque quelque peu de robustesse en ce qui concerne la valeur à attribuer aux différents paramètres. Certaines valeurs ont pour effet d'augmenter de manière significative le temps d'exécution de l'algorithme (lorsque le nombre de cycles et de boucles croît), sans pour autant produire de meilleurs résultats. Aussi, certaines valeurs des paramètres ont pour effet de dégénérer la qualité des solutions fournies de manière sensible. En attribuant par exemple une valeur élevée aux paramètres *Parq0Sel* et *ParAlphaSel*, l'algorithme fournis des solutions beaucoup plus coûteuses que lorsque ces valeurs sont maintenues à un niveau peu élevé (en deçà de 10 % pour le paramètre *Parq0Sel* et de 5 % pour *ParAlphaSel*). Les résultats dégénèrent par ailleurs de manière significative lorsqu'il est attribué une valeur positive aux paramètres *Parq0Att* et *ParAlphaAtt* relatifs à l'affectation. Cette situation peut s'expliquer en raison de l'efficacité des règles mises en œuvre en ce qui concerne l'affectation (*moindre coût* ou *moindre augmentation*), qui sont en partie ignorées par l'intervention d'une démarche aléatoire. L'opportunité de l'utilisation d'une telle démarche aléatoire en ce qui concerne l'affectation est donc compromise (la démarche partiellement aléatoire concernant l'ordre de sélection s'avère cependant efficace si l'on attribue une faible valeur à ses paramètres, comme nous le constatons à l'aide des résultats obtenus). Il serait toutefois possible de tenter de substituer une nouvelle règle non aléatoire au deuxième argument de la règle pseudo aléatoire proportionnelle, afin de déterminer s'il est possible d'obtenir de meilleurs résultats à l'aide d'une telle substitution. Enfin, soulignons que l'utilisation conjointe de

notre technique d'optimisation locale k-opt avec l'heuristique d'OCF n'apporte des bénéfices que lorsqu'elle est mise en œuvre sur des instances comportant une structure de coûts se rapprochant de celle contenue dans les fichiers créés pour le problème d'affectation simple, alors que l'heuristique d'OCF exécutée seule produit de meilleurs résultats lorsque cette structure des coûts est similaire à celle retrouvée dans les fichiers utilisés pour le problème de domiciliation double.

5.3 Recherches futures

Notre implémentation relative au problème de domiciliation double ne tient compte que de l'affectation statique concernant les liens physiques à établir entre chacune des cellules et l'ensemble des commutateurs du réseau. Cet algorithme pourrait être utilisé afin de déterminer la meilleure configuration du réseau, en fonction du nombre d'appels connu ou anticipé à destination de chacune des cellules pour chacune des périodes modélisées de la journée. Il serait ensuite possible de créer un algorithme dynamique tenant compte de ces liens physiques existants, qui aurait pour fonction de déterminer en temps réel les moments opportuns des changements d'affectation concernant la gestion des cellules qui disposent de deux liens physiques. L'algorithme n'aurait comme tâche, que de parcourir à chaque intervalle de temps t l'ensemble des cellules disposant de deux liens afin de déterminer s'il y a lieu d'effectuer un changement d'affectation. Seule l'expérimentation effectuée sur des données réelles pourrait déterminer l'intervalle t le plus efficace, en tenant compte des fréquences de changements des schémas dans le réseau réel et des coûts associés aux transferts des données par chacune des cellules impliquées, concernant le nombre d'appels moyen ayant été reçus au cours de la période précédente. Le changement d'affectation concernant la gestion d'une cellule serait jugé opportun lorsqu'il aurait pour effet de diminuer les coûts de relèves relatifs à la cellule, en se basant sur les coûts engendrés au cours de la période précédente.

Enfin, nous croyons que l'heuristique d'OCF est une méthode relativement flexible et qui peut s'adapter à toutes les situations, comme nous avons pu le démontrer

en l'appliquant à des fichiers contenant des structures de coûts très différentes. Il serait cependant possible d'améliorer davantage son implémentation par rapport à celle effectuée au cours de la présente étude. Il serait notamment intéressant de tenter d'améliorer la partie de l'algorithme chargée de mettre en œuvre la règle pseudo aléatoire proportionnelle concernant l'affectation afin de la rendre plus efficace. Le deuxième argument de la règle, consistant en une recherche purement aléatoire dans notre algorithme, pourrait ainsi être remplacé par une deuxième procédure «déterministe» qui aurait éventuellement pour effet d'améliorer le comportement de la règle, procurant de meilleurs résultats.

Soulignons que l'efficacité de notre algorithme semble augmenter parallèlement avec la taille des instances des problèmes, comparativement aux autres méthodes analysées dans la présente étude. Il serait intéressant de l'appliquer à des instances comprenant un nombre de cellules et de commutateurs plus importants afin de vérifier si cet accroissement se poursuit.

ANNEXE A

Tableau A.1 Résultats comparatifs entre la méthode RT-PC (André et al.) et la variante OCF 1/1/0 et 1/0/2

Nb Cell.~ Nb Comm.		RT/VNS (Sol. Init PC)			Diff (%)	ACO/Kopt "1/0/2" 7/7/95/100/0.1/0/0/5.5/5.5			Diff (%)
30~3	1	296	0.05		0.0327	0.06	0.07	296	0.0000
	10	338	0.03		0.0029	0.06	0.07	338	0.0000
	11	272	0.02		0.0037	0.06	0.07	273	0.0037
	12	298	0.05		0.0230	0.06	0.07	300	0.0067
	13	292	0.05		0.0903	0.07	0.07	320	0.0875
	14	382	0.04		0.0026	0.06	0.07	383	0.0026
	15	346	0.07		0.0142	0.06	0.07	346	0.0000
	16	271	0.03		0.0145	0.06	0.07	275	0.0145
	17	275	0.03		0.0351	0.07	0.07	284	0.0317
	18	316	0.02		0.0000	0.06	0.07	316	0.0000
	19	390	0.04		0.0347	0.07	0.08	400	0.0250
	20	269	0.03		0.0000	0.06	0.07	269	0.0000
	21	304	0.03		0.0033	0.06	0.07	305	0.0033
	22	252	0.04		0.0000	0.06	0.07	252	0.0000
	23	370	0.04		0.0212	0.06	0.07	371	0.0027
		4671			0.0191			4728	0.0121
200~7	1	3930	4.59		0.0081	17.84	18.10	3957	0.0068
	10	2985	2.36		0.0116	17.76	18.03	3006	0.0070
	11	2956	1.48		0.0014	17.80	18.07	2958	0.0007
	12	3285	2.81		0.0079	17.75	18.02	3286	0.0003
	13	3361	2.66		0.0030	17.83	18.10	3368	0.0021
	14	3326	1.34		0.0027	17.86	18.13	3329	0.0009
	15	3021	1.18		0.0046	17.75	18.02	3025	0.0013
	16	3285	3.69		0.0018	17.81	18.07	3288	0.0009
	17	3077	3.16		0.0023	17.82	18.09	3080	0.0010
	18	3054	4.19		0.0255	17.81	18.08	3096	0.0136
	19	3617	5.53		0.0208	17.77	18.04	3693	0.0206
	20	3432	6.51		0.0175	17.85	18.11	3487	0.0158
	21	2982	2.38		0.0145	17.78	18.04	2988	0.0020
	22	3053	4.19		0.0381	17.84	18.10	3106	0.0171
	23	3027	0.77		0.0026	17.83	18.10	3033	0.0020
		48391			0.0109			48701	0.0064

ANNEXE B

Tableau B.1 Résultats comparatifs entre la méthode RT-PC (André et al.) et les variantes OCF 1/1/0 et 1/0/2

Nb Cell.~ Nb Comm.	RT/VNS (Sol. Init PC)			Diff (%)			Diff (%)
30~3 1	296	0.05		0.0295			0.0327
10	338	0.03		0.0029			0.0029
11	272	0.02		0.0037			0.0037
12	298	0.05		0.0100			0.0230
13	292	0.05		0.0611			0.0903
14	382	0.04		0.0026			0.0026
15	346	0.07		0.0029			0.0142
16	271	0.03		0.0073			0.0145
17	275	0.03		0.0283			0.0351
18	316	0.02		0.0000			0.0000
19	390	0.04		0.0323			0.0347
20	269	0.03		0.0000			0.0000
21	304	0.03		0.0033			0.0033
22	252	0.04		0.0156			0.0000
23	370	0.04		0.0289			0.0212
	4671			0.0156			0.0191
100~5 1	1611	0.62		0.0213			0.0037
10	1386	0.30		0.0014			0.0014
11	1166	0.47		0.0348			0.0043
12	1238	0.32		0.0016			0.0024
13	1325	0.37		0.0112			0.0067
14	1201	0.49		0.0066			0.0361
15	1270	0.65		0.0201			0.0473
16	1355	0.60		0.0245			0.0376
17	1173	0.20		0.0000			0.0000
18	1409	0.54		0.0229			0.0098
19	1308	0.19		0.0000			0.0000
20	1255	0.17		0.0016			0.0008
21	1821	0.68		0.0146			0.0189
22	1186	1.05		0.0373			0.0247
23	1223	0.17		0.0000			0.0000
	19927			0.0135			0.0132

Tableau B.2 Résultats comparatifs entre la méthode PC employée seule (André et al.) et les variantes OCF 1/1/0 et 1/0/2

Nb Cell.~ Nb Comm.		PC (Sol. Init)				Diff (%)				Diff (%)
30~3	1	305	0.01			0.0000				0.0033
	10	339	0.02			0.0000				0.0000
	11	273	0.00			0.0000				0.0000
	12	310	0.01			-0.0290				-0.0164
	13	304	0.01			0.0225				0.0530
	14	383	0.01			0.0000				0.0000
	15	347	0.00			0.0000				0.0114
	16	274	0.01			-0.0037				0.0036
	17	283	0.01			0.0000				0.0070
	18	316	0.01			0.0000				0.0000
	19	400	0.00			0.0074				0.0099
	20	269	0.00			0.0000				0.0000
	21	305	0.01			0.0000				0.0000
	22	252	0.01			0.0156				0.0000
	23	381	0.00			0.0000				-0.0079
		4741				0.0006				0.0044
100~5	1	1637	0.05			0.0055				-0.0124
	10	1393	0.07			-0.0036				-0.0036
	11	1203	0.07			0.0041				-0.0273
	12	1244	0.07			-0.0032				-0.0024
	13	1329	0.06			0.0082				0.0037
	14	1207	0.05			0.0017				0.0313
	15	1271	0.06			0.0193				0.0465
	16	1361	0.05			0.0202				0.0334
	17	1174	0.06			-0.0009				-0.0009
	18	1417	0.06			0.0173				0.0042
	19	1308	0.06			0.0000				0.0000
	20	1255	0.06			0.0016				0.0008
	21	1853	0.06			-0.0027				0.0016
	22	1197	0.06			0.0284				0.0156
	23	1223	0.07			0.0000				0.0000
		20072				0.0063				0.0060

Tableau B.3 Résultats comparatifs entre la méthode RT (Houéto et al.) et les variantes OCF 1/1/0 et 1/0/2

Nb Cell.~ Nb Comm.		RT (Houéto)				Diff (%)				Diff (%)
30~3	1	286	0.08			0.0280				0.0327
	10	343	0.18			-0.0116				-0.0118
	11	272	0.10			0.0037				0.0037
	12	299	0.09			0.0066				0.0197
	13	292	0.09			0.0611				0.0903
	14	382	0.20			0.0026				0.0026
	15	346	0.16			0.0029				0.0142
	16	271	0.13			0.0073				0.0145
	17	275	0.11			0.0283				0.0351
	18	316	0.12			0.0000				0.0000
	19	390	0.09			0.0323				0.0347
	20	289	0.08			0.0000				0.0000
	21	304	0.09			0.0033				0.0033
	22	252	0.10			0.0156				0.0000
	23	370	0.12			0.0286				0.0212
		4577				0.0143				0.0178
100~5	1	1611	0.49			0.0213				0.0037
	10	1390	0.62			-0.0014				-0.0014
	11	1167	0.67			0.0339				0.0034
	12	1238	0.51			0.0016				0.0024
	13	1330	1.02			0.0075				0.0030
	14	1203	0.90			0.0050				0.0345
	15	1271	0.74			0.0193				0.0465
	16	1368	0.79			0.0151				0.0284
	17	1173	0.55			0.0000				0.0000
	18	1412	1.00			0.0206				0.0077
	19	1308	0.56			0.0000				0.0000
	20	1255	0.59			0.0016				0.0008
	21	1861	0.76			-0.0070				-0.0027
	22	1186	0.76			0.0373				0.0247
	23	1223	0.51			0.0000				0.0000
		19996				0.0101				0.0096

Tableau B.4 Résultats comparatifs entre la méthode RT avec Chaînes d'éjection (André et al.) et les variantes OCF 1/1/0 et 1/0/2

Nb Cell.~ Nb Comm.		RT (CE)				Diff (%)				Diff (%)
30~3	1	296	0.02			0.0295				0.0327
	10	338	0.02			0.0029				0.0029
	11	272	0.02			0.0037				0.0037
	12	298	0.02			0.0100				0.0230
	13	292	0.02			0.0611				0.0903
	14	382	0.02			0.0026				0.0026
	15	347	0.02			0.0000				0.0114
	16	271	0.02			0.0073				0.0145
	17	275	0.02			0.0283				0.0351
	18	316	0.02			0.0000				0.0000
	19	390	0.01			0.0323				0.0347
	20	269	0.02			0.0000				0.0000
	21	304	0.02			0.0033				0.0033
	22	252	0.01			0.0156				0.0000
	23	370	0.02			0.0289				0.0212
		4672				0.0154				0.0189
100~5	1	1610	0.09			0.0219				0.0043
	10	1386	0.09			0.0014				0.0014
	11	1167	0.11			0.0339				0.0034
	12	1238	0.09			0.0016				0.0024
	13	1325	0.09			0.0112				0.0067
	14	1203	0.10			0.0050				0.0345
	15	1270	0.10			0.0201				0.0473
	16	1358	0.10			0.0223				0.0355
	17	1173	0.09			0.0000				0.0000
	18	1412	0.10			0.0208				0.0077
	19	1308	0.08			0.0000				0.0000
	20	1255	0.10			0.0016				0.0008
	21	1823	0.10			0.0135				0.0178
	22	1185	0.12			0.0381				0.0255
	23	1223	0.10			0.0000				0.0000
		19936				0.0130				0.0127

BIBLIOGRAPHIE

Amoussou G.-E., Pesant G., Pierre S., « Affectation de cellules à des commutateurs par programmation par contraintes », *IEEE CCECE'2001*, May 13-16, 2001, Toronto, Canada.

Amoussou G.-E., André M., Pesant G., Pierre S., « Une approche basée sur la programmation par contraintes pour affecter des cellules à des commutateurs dans les réseaux cellulaires pour mobiles », *Annales des télécommunications*, soumis pour publication en Novembre 2001.

André M., Amoussou G., Pesant G., Pierre S., « An exact constraint programming algorithm for the assignment of cells to switches », *SCRO-JOPT, 43e Congrès annuel de la Société canadienne de recherche opérationnelle (SCRO)*, conjointement avec les *Journées de l'optimisation*, 6 au 9 Mai 2001, Québec, Canada.

André M., Pesant G., Pierre S., « A variable neighborhood search approach for assigning cells to switches in wireless networks », *IEEE Transactions on Systems, Man, and Cybernetics*, soumis pour publication en Mars 2002.

André M., Pesant G., Pierre S., « Affectation de cellules à des commutateurs de réseaux cellulaires mobiles par programmation par contraintes », *IEEE Canadian Conference on Electrical and Computer Engineering*, CCECE'2002, 9-12 May, 2002, Winnipeg, Canada, pp.1311-1316.

André M., Pesant G., Pierre S., « A constraint programming approach for the assignment of cells to switches in a mobile communication network », *Journées de l'optimisation 2002*, 15-17 Mai 2002, Montréal, Canada, p. 32.

André M., Pesant G., Pierre S., « Using tabu search for assigning cells to switches in mobile cellular networks », *21st Biennal Symposium on Communications*, June 2 - 5, 2002, Kingston, Canada, pp. 129-133.

Beckers R., Deneubourg J.L. et Goss S. (1992), « Trails and U-turns in the selection of the shortest path by the ant Lasius niger », *Journal of theoretical biology*, Vol. 159, pp. 397-415.

Beckers R., Deneubourg J.-L. et Goss S., « Modulation of trail laying in the ant *Lasius niger* (hymenoptera: Formicidae) and its role in the collective selection of a food source », *Journal of Insect Behavior*, 1993, 6 (6), pp 751-759.

Bonabeau E. et Théraulaz G., « Swarm smarts », *Scientific American*, March 2000, pp. 72-79.

Brady R. M., « Optimization strategies gleaned from biological evolution », *Nature* October 31, 1985, Vol. 317, pp. 804-806.

Bullnheimer B., Hartl R. F. et Strauss C., « A new rank-based version of the ant system: A computational study », *Central European Journal for Operations Research and Economics*, 1999, 7 (1): pp. 25-38.

Cerny V., « A thermodynamical approach to the travelling salesman problem: An efficient simulation algorithm », *J. Optimization Theory and Appl*, 1985, 45, pp. 41-51.

Den Besten M., Stützle T. et Dorigo M., « Ant colony optimization for the total weighted tardiness problem », in M. Schoenauer, K. Deb, G. Rudolph, X. Yao, E. Lutton, J. J. Merelo, and H.S. Schwefel, editors, *Proceedings of PPSN-VI, Sixth International Conference on Parallel Problem Solving from Nature* of *Lecture Notes in Computer Science*, Springer Verlag, Berlin, Germany, 2000, Vol. 1917, pp. 611-620.

Di Caro G. et Dorigo M., « AntNet: A mobile agents approach to adaptive routing. Technical report », *IRIDIA/97-12*, IRIDIA, Université Libre de Bruxelles, Belgium, 1997.

Di Caro G. et Dorigo M., « AntNet: Distributed stigmergetic control for communications networks », *Journal of Artificial Intelligence Researc*h, 1998a, Vol. 9, pp. 317-365.

Di Caro G. et Dorigo M., « Ant colonies for adaptive routing in packet-switched communications networks », in A. E. Eiben, T. Bäck, M. Schoenauer, and H.-P. Schwefel, editors, *Proceedings of PPSN-V, Fifth International Conference on Parallel Problem Solving from Nature* of *Lecture Notes in Computer Science*, Springer Verlag, Berlin, Germany, 1998b, Vol. 1498, pp. 673-682.

Di Caro G. et Dorigo M., « Mobile agents for adaptive routing », in H. El-Rewini, editor, *Proceedings of the 31st International Conference on System Sciences (HICSS-31), IEEE Computer Society Press*, Los Alamitos, CA, 1998c, pp. 74-83.

Dorigo M., Maniezzo V. et Colorni A., « The ant system: An autocatalytic optimizing process », *Technical Report 91-016 Revised*, Dipartimento di Elettronica, Politecnico di Milano, Italy, 1991a.

Dorigo M., Maniezzo V. et Colorni A., « Positive feedback as a search strategy », *Technical Report 91-016*, Dipartimento di Elettronica, Politecnico di Milano, Italy, 1991b.

Dorigo M., « Optimization, learning and natural algorithms » (in Italian), PhD thesis, Dipartimento di Elettronica, Politecnico di Milano, Italy, 1992, p. 140.

Dorigo M., Maniezzo V. et Colorni A., « The ant system: optimization by a colony of cooperating agents », *IEEE Transactions on Systems, Man, and Cybernetics – Part* B, 1996, 26 (1), pp. 29-41.

Dorigo M. et Gambardella L.M., « Ant colony system: A cooperative learning approach to the traveling salesman problem », *TR/IRIDIA/1996-5*, Université Libre de Bruxelles et *IEEE Transactions on Evolutionary Computation*, 1997, Vol.1, No.1.

Dorigo M. et Gambardella L. M., « Ant colonies for the traveling salesman problem », *BioSystems*, 1997a, Vol. 43, pp. 73-81.

Dorigo M. Gambardella L. M., « Ant colony system: A cooperative learning approach to the traveling salesman problem », *IEEE Transactions on Evolutionary Computation*, 1997b, 1 (1), pp. 53-66.

Dorigo M., Di Caro G. et Gambardella L. M., « Ant algorithms for discrete optimization », *Artificial Life*, 1999, 5 (2), pp. 137-172.

Dorigo M. et Di Caro G., « The ant colony optimization meta-heuristic », in D. Corne, M. Dorigo, and F. Glover, editors, *New Ideas in Optimization*, McGraw Hill, London, UK, 1999, pp. 11-32.

Freisleben B. et Merz P., « Genetic local search algorithm for solving symmetric and asymmetric traveling salesman problems », *Proceedings of IEEE International Conference on Evolutionary Computation*, 1996a, *IEEE-EC 96*, IEEE Press, pp. 616-621.

Freisleben B. et Merz P., « New genetic local search operators for the traveling salesman problem », *Proceedings of PPSN IV–Fourth International Conference on Parallel Problem Solving From Nature*, H.–M. Voigt, W. Ebeling, I. Rechenberg and H.–S. Schwefel (Eds.), Springer-Verlag, Berlin, 1996b, pp. 890-899.

Gambardella L. M. et Dorigo M., « Solving symmetric and asymmetric TSPs by ant colonies », in *Proceedings of the 1996 IEEE International Conference on Evolutionary Computation (ICEC'96)*, *IEEE Press*, Pis-cataway, NJ, 1996, pp. 622-627.

Gambardella L. M., Taillard E. D. et Agazzi G., « A multiple ant colony system for vehicle routing problems with time windows », MACS-VRPTW, in D. Corne, M. Dorigo, and F. Glover, editors, *New Ideas in Optimization*, McGraw Hill, London, UK, 1999, pp. 63-76.

Gambardella L. M. et Dorigo M., « Ant Colony System hybridized with a new local search for the sequential ordering problem », *INFORMS Journal on Computing*, 2000, 12 (3), pp. 237-255.

Glover F., « Tabu search – Part 1 », *ORSA Journal on Computing*, 1989, Vol. 1, No.3, pp. 190-206.

Glover F., « Tabu search – Part 2 », *ORSA Journal on Computing*, 1990b, Vol. 2, pp. 4-32.

Glover F., « Tabu search: A tutorial », *INTERFACES*, 1990a, Vol. 20, No.4, pp. 74-94.

Goss. S., Aron. S., Deneubourg J.L. et J.M. Pasteels, « Self-organized shortcuts in the argentine ant », *Naturwissenschaften*, 1989, Vol. 76, pp. 579-581.

Hedible C. et Pierre S., « Algorithme génétique pour l'affectation des cellules à des commutateurs dans les réseaux de communications personnelles », in *Proceedings of the IEEE CCECE'2000*, May 6-10, 2000, Halifax, Canada, pp. 1077-1081.

Hedible C. et Pierre S., « Une approche heuristique d'affectation de cellules à des commutateurs dans les réseaux cellulaires », *Journées de l'optimisation 2000*, 15-17 Mai 2000, Montréal, Canada, p. 31.

Hedible C. et Pierre S., « An evolutionary method for assigning cells to switches in cellular networks », *IEEE Transactions on Systems, Man, and Cybernetics*, soumis pour publication en Avril 2002.

Hedible C. et Pierre S., « A genetic algorithm for assigning cells to switches in personal communication networks », *IEEE Transactions on Systems, Man and Cybernetics – Part A*, soumis pour publication en juillet 2002.

Hoffmeyer J., « The swarming body », *Proc. 5th Congress of the International Association for Semiotic Studies*, Berkeley, 1994.

Holland J. H. « Genetic algorithms and the optimal allocation of trials », *SIAM Journal of Computing*, 1973, Vol.2, No.2, pp. 88-105.

Holland J. H. « Adaptation in natural and artificial systems », *The University of Michigan Press*, Ann Arbor, Michigan 1975.

Houéto F. et Pierre S., « Affectation de cellules à des commutateurs dans les réseaux de communications personnelles », in *Proceedings of the IEEE CCECE'2000*, May 6-10, 2000, Halifax, Canada, pp. 1037-1041.

Houéto F. et Pierre S., « Affectation des cellules à des commutateurs dans les réseaux de communications personnelles », *Journées de l'optimisation 2000*, 15-17 Mai 2000, Montréal, Canada, p. 32.

Houéto F. et Pierre S., « Assigning cells to switches using tabu search », *The Twelfth Annual International Conference on Wireless Communications, Wireless 2000*, July 10-12, 2000, Calgary, Canada, pp. 438-447.

Houéto F. et Pierre S., « Affectation heuristique de cellules à des commutateurs dans les réseaux cellulaires mobiles », *Annales des Télécommunications*, 2001, Vol. 56, Nos. 3-4, pp. 184-197.

Houéto F. et Pierre S., « A tabu-search approach for assigning cells to switches in cellular mobile networks », *Computer Communications*, Mars 2002, Vol. 25, No. 5, pp. 465-478.

Houéto F. et Pierre S., « Assigning cells to switches in cellular mobile networks using tabu search », *IEEE Transactions on Systems, Man, and Cybernetics: Part B (Cybernetics)*, June 2002, Vol. 32: Part B, No. 3, pp. 351-356.

Jaffar J. et Lassez J.L., « Constraint logic programming », *In Proceedings of the 14th ACM Symposium on Principles of Programming Languages*, Munich, Germany, ACM Press, January 1987, pp. 111-119.

Johnson D.S. et McGeoch L.A., « The travelling salesman problem: a case study in local optimization », en cours de publication in *Local Search in Combinatorial Optimization*, E.H.L. Aarts and J.K. Lenstra (Eds.), Wiley and Sons: New York.

Kirkpatrick S., « Optimization by simulated annealing: Quantitative studies », *J. Stat. Physics*, 1984, Vol. 34, pp. 976-986.

Lam J. et Delosme J.-M., « An efficient simulated annealing schedule: implementation and evaluation », 1988, manuscript.

Lin S. et Kernighan B.W., « An effective heuristic algorithm for the traveling-salesman problem », *Oper. Res.*, 1973, Vol. 21, pp. 498-516.

Martin O., Otto S. W. et Felten E. W., « Large-step markov chains for the traveling salesman problem », *Complex Systems*, 1991, Vol. 5, pp. 299-326.

Merchant A. Sengupta B., NEC C & C Research Laboratories, « Multiway graph partitioning with applications to PCS networks », *IEEE Infocom 1994*, 1994, Vol.2, pp. 593-600.

Merchant A. et Sengupta B., « Assignment of cells to switches in PCS networks », *IEEE/ACM Transactions on Netorking*, October 1995, Vol. 3, No. 5, pp. 521-526.

Merkle D., Middendorf M. et Schmeck H., « Ant colony optimization for resource-constrained project scheduling », in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2000)*, Morgan Kaufmann Publishers, San Francisco, CA, 2000, pp. 893-900.

Muhlenbein H., Gorges-Schleuter M. et Kramer O., « Evolution algorithms in combinatorial optimization », *Parallel Compu.*, 1988, Vol. 7, pp. 65-85.

Reinelt G., « The traveling salesman: computational solutions for TSP applications », Springer-Verlag, 1994.

Selman B. of Dept. of Computer Science, Cornall University, http://www.cs.cornell.edu/home/selman/papers-ftp/98.mitecs.greedy.ps.

Skorin-Kapov J., « Tabu search applied to the quadratic assignment problem », *ORSA Journal on Computing*, 1989, Vol. 2, No.1, pp. 33-45.

Stützle T. et Hoos H. H., « The ant system and local search for the traveling salesman problem », in T. Bäck, Z. Michalewicz and X. Yao, editors, *Proceedings of the 1997 IEEE International Conference on Evolutionary Computation (ICEC'97), IEEE Press*, Piscataway, NJ, 1997, pp. 309-314.

Stützle T. et Dorigo M., « ACO algorithms for the quadratic assignment problem », in D. Corne, M. Dorigo and F. Glover, editors, *New Ideas in Optimiza-tion*, McGraw Hill, London, UK, 1999, pp. 33-50.

Stützle T., « Local search algorithms for combinatorial problems: Analysis, improvements and new applications », Infix, Sankt Augustin, Germany, 1999.

Stützle T. et Hoos H. H., « Ant system. *Future generation computer systems* », 2000, 16 (8), pp. 889-914.

Sutherland I., « Skethpad, a man-machine graphical communication system », *in Proceedings of the Spring Joint Computer Conference*, IFIPS, 1963, pp. 329-346.

Sutton R. S. et Barto A. G., « Reinforcement learning: An introduction », *MIT Press*, Cambridge, MA, 1998.

Van Hentenryck P., Deville Y. et Michel L., « Numerica, a modeling language for global optimization », *MIT Press*, 1997.

Ward M., « There's an ant in my phone », *New Scientist*, 24 January 1998.

AUTRES RÉFÉRENCES

Les figures 1.1 et 1.2 proviennent du site Internet de l'*International Engineering Consortium* situé à l'adresse suivante : http://www.iec.org/

Le développement relatif aux principes des algorithmes génétique a été rédigé à l'aide des informations contenues sur le site Internet situé à l'adresse suivante : http://cs.felk.cvut.cz/~xobitko/ga/

L'exemple des Figures 2.1 et 2.2 provient de ce même site Internet situé à l'adresse suivante : http://cs.felk.cvut.cz/~xobitko/ga/

Le développement relatif aux principes de la recherche taboue a été rédigé à l'aide des informations contenues sur les sites Internet situés aux adresses suivantes : http://www.winforms.phil.tu-bs.de/winforms/research/tabu/tabu.html, http://www.cs.umass.edu/~young/grad/tardy/section2_7_7.html#SECTION00770000000000000000 et http://www-users.cs.york.ac.uk/~mark/year1/chapter2_5.html#SECTION00500000000000000000 ainsi qu'aux liens auxquels ils référent.

Le développement relatif aux principes de la programmation par contrainte a été rédigé à l'aide d'informations contenues sur les sites Internet situés aux adresses suivantes : http://www.icparc.ic.ac.uk/eclipse/reports/handbook/handbook.html et http://ktiml.mff.cuni.cz/~bartak/constraints/, ainsi qu'aux liens auxquels ils référent.

Les Sections 3.1 et 3.3 ont été rédigée en ayant recours aux informations contenues dans les articles et les sites Internet suivants :

Dorigo M. et Di Caro G., « The ant colony optimization meta-heuristic », *IRIDIA*, Université Libre de Bruxelles, à paraître dans D. Corne, M. Dorigo and F. Glover, editors, *New Ideas in Optimization*, McGraww-Hill, 1999.

Dorigo M., « The ant colony optimization metaheuristic : Algorithms, applications and advances », Université Libre de Bruxelles, *IRIDIA*; Stützle T., TU Darmstadt, Computer Science, Intellectics Group, Germany.

Dorigo M. et Di Caro G., « Ant algorithms for discrete optimization », Massachusetts Institute of Technology, Artificial Life, 1999, 5, pp. 137-172.

Stützle T., Université Libre de Bruxelles, IRIDIA, Brussels, Belgium, Hoos H. H., University of British Columbia, Department of Computer Science, Vancouver, Canada, « MAX-MIN Ant System », soumis pour publication à *Elsevier* Science le 5 novembre 1999.

Dorigo M., Gambardella L. M., « Ant colony system: A cooperative learning approach to the traveling salesman problem », TR/IRIDIA/1996-5, Université Libre de Bruxelles, Belgium, accepté pour publication dans *IEEE Transactions on Evolutioary Computation*, Vol. 1, No. 1, 1997.

Gambardella L. M., Dorigo M., « An ant colony system hybridized with a new local search for the sequential ordering problem », accepté pour publication dans *Informs Journal on Computing*.

Swarm Intelligence : http://www.particleswarm.net/papers.html et
http://www.engr.iupui.edu/~shi/Coference/psopap4.html#abs
ainsi qu'aux liens auxquels ils réfèrent.

Ainsi que les articles référencés dans la bibliographie.

La Section 3.2 a été rédigée en ayant recours aux informations contenues dans les articles suivants :

Helsgaun K., « An effective implementation of the Lin-Kernighan traveling salesman heuristic », *Department of Computer Science*, Roskilde University, Denmark.

Lin S. et Kernighan B.W., « An effective heuristic algorithm for the traveling-salesman problem », *Oper. Res.*, 1973, Vol. 21, pp. 498-516.

Ainsi que les informations contenues sur le site situées à l'adresse suivante : http://www.cs.cornell.edu/home/selman/papers-ftp/98.mitecs.greedy.ps.

Les Figures 3.1 et 3.2 proviennent de l'article « Swarm smarts », *Scientific American*, Bonabeau E. et Théraulaz G., March 2000, pp. 72-79.

Les Figures 3.4 et 3.5 proviennent de l'article « An effective implementation of the Lin-Kernighan traveling salesman heuristic », Helsgaun K., *Department of Computer Science*, Roskilde University, Denmark.

Les Figures 3.6 à 3.9 proviennent du site Internet situé à l'adresse suivante : http://iridia.ulb.ac.be/~mdorigo/ACO/RealAnts.html